AD-A275 848

A/GATECH DOSP FABRICATION

University of Alabama in Huntsville

Dr. H.J. Caulfield, Steven Kupiec

DTIC
ELECTE
FEB 16 1994
S
B
D

DTIC QUALITY INSPECTED 2

**Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York**

94   2   15   086

94-05100

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-93-224 has been reviewed and is approved for publication.

APPROVED:

DAVID R. CHRISTIE, 1Lt, USAF
Project Engineer

FOR THE COMMANDER:

DONALD W. HANSON
Director of Surveillance & Photonics

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE November 1993	3. REPORT TYPE AND DATES COVERED Final Apr 91 -- Apr 92

4. TITLE AND SUBTITLE
A/GATECH DOSP FABRICATION

5. FUNDING NUMBERS
C - F30602-88-D-0025, Task 0048
PE - 62702F
PR - 4600
TA - P3
WU - PA

6. AUTHOR(S)
Dr. H.J. Caulfield, Steven Kupiec

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Alabama in Huntsville
Center for Applied Optics
Huntsville AL 35899

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (OCPB)
25 Electronic Pky
Griffiss AFB NY 13441-4515

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-93-224

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: David R. Christie, 1Lt, USAF/OCPB/(315) 330-7670
Prime Contractor: Georgia Institute of Technology, Atlanta GA 30332

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
The beginning of a true massively parallel processor based on holographic technology is presented. The theory, algorithms, and initial research are set forth and explored. Preliminary research results show great possibility for this architecture in areas of large-scale interconnects, and low-energy processing. Future research efforts, as well as possible applications, are also discussed.

14. SUBJECT TERMS
Digital, Optical, Processor, Holographic, Parallel, Low-energy, Interconnection

15. NUMBER OF PAGES
112

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

TABLE OF CONTENTS

Chapter 1

The HOPLA Architecture

1.1.Introduction:

The general properties of a new digital optical computer architecture, the optical programmable logic array (HOPLA), are discussed. The advantages such as immense parallelism and sub-kT operation of the HOPLA are discussed. Problems deriving from limitations on logical complexity and on fixed interconnection are addressed. It is shown that the architecture may be used for general purpose operation with the use of pre- and post-processors.

1.1.1 Objective:

In this chapter we describe a new architecture: the Optical Programmable Logic Array (HOPLA) based upon the N^4 interconnect devised by Caulfield [1,2]. It is capable of implementing arbitrary logic by the use of the control operator method proposed by Morozov [3] and elaborated upon by Guilfoyle[4,5]. It is functionally identical to the class of electronic devices known as Programmable Logic Arrays (PLAs) although on a considerably larger scale. Unlike previous designs implementing the control operator method, the HOPLA is a Wave Particle Duality (WPD) feasible processor[6,7,8] fufiling both the physical and computational criteria for WPD operation, and thus capable of operating at exceedingly low power consumption levels.

In addition we shall address the issue of generating the large ($\sim 10^{5-6}$) fan-ins necessary for achieving sub-kT efficiency within a WPD capable processor. We shall also illustrate that although the logic is fixed within our processor, our design is capable of being reprogramed by a combination of flexible pre- and post- processors as well as by the use of stored programs.

1.1.2. Background:

Within the past decade it has become a tenet of optical computing that for an optical computer to be feasible, its performance, relative to that of a comparable electronic system, must be qualitatively superior, given that electronics is the established and mature technology. One considerable advantage to optical systems is the capacity for three dimensional interconnectivity. Guilfoyle [4] has suggested that the optical computer should utilize a "global parallelism" which not only executes large number of operations in parallel but interrelates the input data in such a fashion that the output is dependent upon all of the input data. Such "global" parallelism is essentially the digital equivalent of neural networks. But high interconnectivity has an importance beyond that of simple utility. Caulfield et al. [6,7,8] have noted that certain configurations of highly interconnected optical computers comprise Wave Particle Duality (WPD) computers. Such WPD computers are theoretically capable of operating at an energy consumption rate of less than kT per operation, where k is the Boltzmann constant and T is the absolute temperature. In contrast, current electronic digital computers operate at energy consumption rates on the order of 10^6-10^{10} kT per operation.

There exist a broad range of designs for implementing digital operations via optics. Both Cathey et al.[9] and Mirsalehi et al.[10] have produced extensive reviews of the topic. Schemes which involve high degrees of parallelism include the Optical Parallel Array Logic System (OPALS) architecture[11], symbolic substitution archiect-ures[12], look-up table architectures[13], and various forms of the control operator method[3,4,5]. As we will see, the control operator method subsumes table look-up methods and has many similarities to the OPALS architecture. The unique advantages of the control operator method are that it is capable of exploiting large fan-ins as well as only requiring a single threshold level in the interpretation of its results.

1.1.3 The Control Operator Method:

The control operator method was devised by Morozov [3] as a means of implementing digital logic on computers via the use of matrix multiplication operations. The control operator technique has several unique advantages such as generaity of operations, "global" parallelism, and, simple implementation.

Optical systems are well suited for analog matrix multiplication. There exist a broad range of methods for implementing such operations[14]. By constraining the values within our matrices and vectors to either 0 or 1 and thresholding the results, it is possible to define a form of logical matrix operation we shall refer to as a boolean matrix. Since the operations involving boolean matrices are

simply thresholded analog operations, any system capable of achieving analog matrix operations may be converted to a boolean matrix system in a simple manner as shown below:

$$
\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
$$

Thus in terms of logical equations after thresholding:

$$
\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} a \\ a' \\ b \\ b' \\ c \\ c' \end{bmatrix} = \begin{bmatrix} a+b+c \\ a'+c \\ a'+b'+c' \end{bmatrix}
$$

When arithmetic is thus constrained to the values of 1 (true) or 0 (false) we discover that the addition and multiplication operations become the OR (+) and AND (*) operations respectively. Thus Boolean matrices follow the same rules for matrix multiplication as conventional matrices but with the substitution of the AND operation for multiplication and the OR operation for addition. Note that the control operator technique always uses bright logic where 1 denotes a true value, the presence of light, or, a clear or reflective pixel in the case of an SLM.

Examining the results of such matrices operating upon a vector of binary variables we find that the result is a boolean vector. The components of this resulting vector are obtained by the ORing of the various elements of the input vector. Unfortunately the logical OR operator is not sufficient to form a complete system of logic. That is no combination of ORs is capable of generating an AND or a NOT(')[15]. Fortunately it is possible to generate the AND operator from the OR operator by the use of the NOT operator. This is possible due to DeMorgan's laws (i.e. $(ab'c)'=a'+b+c'$ and $(a+b'+c)'=a'bc'$)[15]. Thus the combination of the OR and NOT operator may be used to form a complete system of logic in which it is always possible to reduce any logical function to the sum (series of ORs) of a number of products (series of ANDs) of logical variables and their complements [15]. The format of such an equation is referred to as the Sum Of Products (SOP) or disjunctive normal form. Such a normal form serves as a useful intermediate form for the transformation of of a logical function into matrix form.

We now consider the means by which an arbitrary set of logical functions may be decomposed into a pair of boolean matrix operations. To facilitate our discussion we offer a set of functions which are used in section 4.2 of Reference [16]. The truth tables of these functions are shown in figure 1.1.

The disjunctive normal form of a function may immediately be constructed by examination of the truth table. Each term in this form contains all of the input variables or their complements: i.e. $x_1 x_2 x'_3$ corresponds to the 1 1 0 row of this truth table and is only true for that row. The disjunctive normal form is thus simply constructed by ORing together the terms corresponding to rows in which the function is true (1). In the case of our example:

$$
z_1(x_1,x_2,x_3) = x'_1 x'_2 x'_3 + x_1 x'_2 x'_3 + x_1 x'_2 x_3 \qquad [3]
$$
$$
+ x_1 x_2 x_3
$$

$$
z_2(x_1,x_2,x_3) = x'_1 x'_2 x_3 + x'_1 x_2 x'_3 + x'_1 x_2 x_3 \qquad [4]
$$
$$
+ x_1 x_2 x'_3
$$

$$z_3(x_1,x_2,x_3)= x'_1x'_2x'_3 + x'_1x'_2x_3 + x'_1x_2x'_3 \quad [5]$$
$$+ x_1x'_2x_3 + x_1x_2x'_3 + x_1x_2x_3$$

The disjunctive normal form, although theoretically useful, is in most cases not the simplest expression of a logical function. As noted above it is possible to minimize a logical function by the use of several methods. The best known of these being the Quine-McCluskey algorithm[15], individually minimizing our functions we obtain:

$$z_1(x_1,x_2,x_3)= x'_2x'_3 + x_1x_3 \quad [6]$$

$$z_2(x_1,x_2,x_3)= x'_1x_3 + x_2x'_3 \quad [7]$$

$$z_3(x_1,x_2,x_3)= x'_2x_3 + x_1x_2 + x'_1x'_3 \quad [8]$$

Such individual minimization techniques may lead to less than optimal results with respect to the entire expression. Since we generate the component terms before we combine them, considerable overall simplification may be achieved by discovering common terms between functions. Several techniques have been devised which allow us to use methods such as the Quine-McCluskey algorithm to optimize for overall simplicity; the Muller method [16] and the tag method [16] are the most notable. In their current form the functions in our example have no common terms. Simultaneously optimizing for greatest common simplicity we obtain:

$$z_1(x_1,x_2,x_3)= x'_2x'_3 + x_1x_3 \quad [9]$$

$$z_2(x_1,x_2,x_3)= x'_1x_3 + x_2x'_3 \quad [10]$$

$$z_3(x_1,x_2,x_3)= x_2x'_3 + x'_1x'_2 + x_1x_3 \quad [11]$$

With this new formulation we now have several common terms, thus minimizing our overall complexity.

Once the optimum overall formulation of the functions have been determined, the functions must then be expressed in the form of two matrix operations interspersed with two logical complements in the following manner:

$$z=S(Px')' \quad [12]$$

where x is a vector of the input variables, z is the vector of resulting functions, and S and P are both matrices. A table is first constructed which lists the terms in relation to their component variables and the functions in which they are members. In our current example such a table is shown in figure 1.2. This table form is of considerable utility, not only for the construction of the desired matrices S and P, but also for the light it sheds on the relationship between the critical components of the system. In the case of both current electronic PLA design and existing optical architectures, this table directly corresponds to the masking or interconnect pattern used, be the medium a matrix of wires or a transmission pattern on an SLM. Due to this direct relationship, two critical parameters of this table its height (which corresponds to the number of terms required for implementation), and its width (which corresponds to the number of inputs and outputs), are used to describe the system. In general these parameters are simply referred to as "height" and "width". Finally, this table may be viewed as a form of schematic diagram of a system allowing one to trace out the operation of the system under differing inputs.

Once the above table has been formed, the S and P matrices may be constructed. The P matrix converts, in concert with the logical complementation operators, the x vector to the constituent terms of the system. As noted above, boolean matrix operators may only form selective OR operations; and thus the generation of the terms requires an indirect method using DeMorgan's laws. First the P vector is constructed such that in operating upon the x vector, it generates the components of the terms ORed together. From our example:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x'_1 \\ x_2 \\ x'_2 \\ x_3 \\ x'_3 \end{bmatrix} = \begin{bmatrix} x'_1 + x'_2 \\ x'_1 + x_3 \\ x_1 + x_3 \\ x_2 + x'_3 \\ x'_2 + x'_3 \end{bmatrix}$$

We then complement x (in reality we simply interchange x_n and x'_n) and complement the result which by DeMorgan's laws generates the terms. Following our example:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x'_1 \\ x_1 \\ x'_2 \\ x_2 \\ x'_3 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ x_1 + x'_3 \\ x'_1 + x'_3 \\ x'_2 + x_3 \\ x_2 + x_3 \end{bmatrix}$$

and:

$$(P\underline{x}')' = \begin{bmatrix} x_1 + x_2 \\ x_1 + x'_3 \\ x'_1 + x'_3 \\ x'_2 + x_3 \\ x_2 + x_3 \end{bmatrix}' = \begin{bmatrix} x'_1 x'_2 \\ x'_1 x_3 \\ x_1 x_3 \\ x_2 x'_3 \\ x'_2 x'_3 \end{bmatrix}$$

Once the necessary terms have been generated in the form of a vector, all that remains is to generate the set of functions by ORing the values of the terms together. Since this is the primary function of the boolean matrix the construction of the S matrix is quite trivial and essentially amounts to a transposed form of the second section of the table we have constructed. In the case of our example:

$$S(P\underline{x}')' = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} x'_1 x'_2 \\ x'_1 x_3 \\ x_1 x_3 \\ x_2 x'_3 \\ x'_2 x'_3 \end{bmatrix}$$

$$= \begin{bmatrix} x'_2 x'_3 + x_1 x_3 \\ x'_1 x_3 + x_2 x'_3 \\ x_2 x'_3 + x'_1 x'_2 + x_1 x_3 \end{bmatrix}$$

$$= \begin{bmatrix} z_1(x_1, x_2, x_3) \\ z_2(x_1, x_2, x_3) \\ z_3(x_1, x_2, x_3) \end{bmatrix} = \underline{z}$$

Thus we have shown how it is possible to express any set of logical functions with two boolean matrix operations and an intermediate logical complementation. The combination of these operators may be expressed as a single operator:

$$\Phi \underline{x} = S(P\underline{x}')' = z \qquad [19]$$

This control operator is equivalent to any set of combinational logic functions, that is, any logical expression which is not dependent upon memory or feedback. It is well known that arbitrary finite automata may be generated by feeding back the outputs of a system of combinatoral logic. Morozov[3] has indicated that such feedback may be expressed as progressive application of the appropriate control operator Φ assuming the output vector z is of the same format as the input vector $\underline{x}$. Thus we have a practical method of implementing arbitrary logic and by feedback, arbitrary finite automata via control operators.

1.1.4 Previous Designs:

As seen in the previous section, the control operator method provides a means of expressing logical expressions as a series of thresholded analog matrix operations. The existence of such a method opens the possibility of using any of the wide range of optical matrix multiplication schemes coupled with thesholding and inversion components, either optical or electronic, to achieve digital logic. Two previous general purpose architectures have been proposed, one by Morozov[3] and one by Guilfoyle[5]. The architectures are similar in their capacity for reconfiguration and their use of SLMs for the agent of interconnection.

Morozov [3] has proposed several designs using the control operator method, the most general architecture being that illustrated in figure 1.3. This system basically consists of two SLM based matrix multipliers with a one dimensional optically addressed SLM configured to operate in a threshold and invert mode. Input is achieved via the use of a one dimensional LED array and output is achieved by the use of a one dimensional photodetector array with electronic thresholding. Programming is achieved by loading the P matrix upon the first SLM and the S matrix upon the second SLM. This system provides for high speed due to the one dimensional nature of all of the data carrying components. By changing the S and P matrices the logic of the system may be reconfigured. The single major limitation of this method is that due to the use of SLMs as the control for interconnection, the input is limited to one dimension, thus limiting the parallelism of the architecture.

Guilfoyle[5] has proposed a radically different system that is capable of achieving two dimensional data input by a systolic method. The Guilfoyle architecture displayed in figure 1.4 uses a two dimensional SLM to input data, shifting each input vector to the adjacent row of pixels. The P matrix is loaded upon the second, stationary SLM. At each cycle the two SLMs are imaged together and the result is focused column-wise onto a one dimensional detector array. The detector array utilizes custom electronics to threshold and invert the results then OR the results with its previous state and then transfer the result to the adjacent detector, thus electronically "following" the motion of the input vector. This system does not use an explicit S vector but instead ORs all terms in the P vector to generate a single, desired logic function. This scheme essentially pipelines progressive matrix-vector products. Since the data input SLM is an acousto-optic device, the row to row transfer is automatic and rapid. The rapid speed of operation coupled with high throughput results in impressive preformance and parallelism, despite the essentially serial output. Since, once again, a SLM mediates the interconnection, the system is capable of reconfiguration. In addition, the dynamic nature of the system allows especially large P matrices to be evaluated in several passes, giving the system an essentially infinite virtual "height", as defined in the previous section. Although the systolic approach allows for two dimensional input, the use of a SLM for the control of the interconnect leads to the limitation of the output array to one dimension. In fact, the requirement of the detector to electronically "follow" the input vector limits the effective detector dimension to a single bit, although the systolic nature of the output array assures that this is a fast serial output.

1.2. The HOPLA

The Optical Programable Logic Array is a new implementation of the control operator method that differs from previous designs in several critical aspects. Input and output are both two dimensional and all elements of the input may interact with all others. The interconnections are not mediated by SLMs but by holographic means. The interconnections are fixed and not reprogrammable. And most importantly the HOPLA is WPD-Capable, and thus capable, under proper conditions, of operating at energy consumption rates of less than kT per operation. This renders a proportional reduction in heat dissipation[6,7,8]. The hardware configuration, WPD properties, and basic operation as a special purpose processor are detailed in a previous paper[17].

As with all control operator method based computers, the HOPLA consists of a pair of matrix-vector multipliers with an intermediate thresholding optical inverter. The optical layout of the system is shown in figure 1.5. In the case of the HOPLA the matrix-vector multipliers consist of a pair of N^4 interconnects. The thresholding inverter is a two dimensional optically-addressed SLM with contrast hard-clipping and inverting features. Input is mediated by an electronically addressed SLM in the first N^4 interconnect; while output is achieved by the use of a two dimensional detector array at the output plane of the second N^4 interconnect. Feedback and storage are achieved by electronic means.

The N^4 interconnect, devised by Caulfield[1,2], serves as the heart of of the HOPLA. The N^4 interconnect functionally operates as a weighted total interconnect between a SLM and an output plane. The interconnect preforms the tensor product between a four-tensor $T_{i,j,k,l}$ stored in the hologram and a matrix $a_{i,j}$ stored in the SLM resulting in a second matrix $b_{k,l}$ as calculated in the following manner:

$$b_{k,l} = \sum_i \sum_j T_{i,j,k,l} b_{i,j} \qquad [20]$$

For purposes of the control operator method we may simply rearrange the input vector x into the two-dimensional $a_{i,j}$ matrix and in the same fashion view the $T_{i,j,k,l}$ tensor as a very large P matrix. The output product is folded into two dimensions in the same fashion as x vector was. The thresholding inverter then complements the intermediate result; and the process is repeated in the same fashion for the S matrix. With the exception of the trivial folding of the input and output vectors, the procedure is identical to the standard control operator method.

In order to grasp the differences between the HOPLA and previous SLM based approaches it is useful to follow the P matrix multiplication through the system for a single bit of the result. A functional diagram of the N^4 interconnect is given in figure 1.6. The x vector is electronically loaded onto the SLM. The hologram array is then illuminated. The hologram array consists of an NxN array of seperate holograms resembling a Page Oriented Holographic Memory (POHM) each projecting a different mask pattern onto the SLM, and each projection possessing a different angular frequency. In the case of the specific output bit under consideration, the hologram at the conjugate point of the bit projects an image corresponding to a row from the P vector which has been folded in the same fashion as the x vector. The mask encounters the SLM and is selectively transmitted evaluating the AND operations of the boolean matrix product. The resulting wavefront is then focused to a point on the output plane executing the OR operations of the boolean matrix product. If light is detected, the result is true. Essentially the result is false if the projected image is completely blocked by the SLM and true otherwise. This is the only information derived from the operation. The results are then detected, tresholded, inverted, and displayed by the optically addressed SLM which serves as the thresholding inverter.

Operationally, the dependence upon the mere presence or absence of light is the greatest strength of the control operator method. Only a single threshold value is required; and the value of the threshold is purely dependent upon the crosstalk and scatter of the N^4 interconnect. The potential of each bit to be simultaneously dependent upon any of the input values allows for values of fan-in suffcent to allow for WPD- feasible operation.

1.2.1 The Height Problem

The optical strengths and weaknesses of the components of the HOPLA, especially the N^4 interconnect, have been extensively discussed[2] but the switching theoretical feature have not been yet considered. In this section we examine the advantages and drawbacks of the HOPLA in terms of

computational complexity. In the next section we will offer possible remedies for the failings of the HOPLA.

An HOPLA capable of operation as a WPD processor would generally incorporate roughly 10^6 inputs organized as a 1024x1024 matrix. The number of possible outputs for a WPD processor is of a comparable order. It would be blatantly impossible to claim that an HOPLA or any other physically realizable device would be capable of implementing any "arbitrary" logical function with such a "width" or number of inputs and outputs. The reason for this is the quality of "height", or the number of intermediate terms, as defined in section 1.3 of this chapter. The number of possible terms for n variables is proportional to 2^n, thus the height should grow exponentially with the width in order to retain generality. But in the case of the HOPLA the height is proportional to the width, since the intermediate result is processed and stored on the thresholding inverter. This is due to the fact that the scale of the thresholding inverter is limited to essentially the same scale as the input.

The second computational difficulty with the design of the HOPLA is the fixed nature of the interconnections. Since the interconnections are mediated by the use of holograms rather than SLMs as in the case of previous designs, the interconnections are incapable of reconfiguration during computation. Although of little importance within special purpose applications of this architecture[17], the fixed nature of the interconnects complicates the task of designing a general purpose processor. In the flexible interconnect systems the "program" resides in the interconnections and the "data" resides within the input vector. Such a division of labor existed within the earliest electronic computers as well; but as programs became more and more complex, this method became infeasible. In current electronic computers the interconnections of the components are fixed and the "program" resides in memory in the same fashion as data. The same "stored program" concept may be used to implement a general purpose architecture on the HOPLA. Since the total possible number of interconnections within an HOPLA system would be on the order of $\sim 2 \times 10^{12}$, the net complexity of the system would seem to greatly surpass that of most microprocessors. Unfortunately the height limitation may considerably limit the number of interconnections to several orders of magnitude less than 10^{12}, on the other hand with a width of 10^6 it is almost certain that an interconnection level would be roughly between 10^7 and 10^9. This complexity level is comparable with current microprocessor levels; and should still be suffcient for stored program operations but may introduce constraints upon the extent of possible global operations.

1.2.2. Examples

The limit on height, for example, is the reascn that we may not add tens of thousands of conventional binary numbers simultaneously in a single cycle. The functions defining the resulting sum would double in complexity with each carry bit. The exponential increase required for each additional bit would overwhelm the height of any conceivable system. The conventional method used by electronics is to shift to multiple cycles and feedback the carry bit. A second approach more common to optical systems is to shift to alternative number systems which avoid carries entirely. Although either approach "solves" the height problem, both approaches also compartmentalize the logical operations, limiting both the complexity of the functions and the fan-in values. But it is the capacity to handle large fan-ins that gives the HOPLA its advantages both in parallelism and WPD operation. Thus in order to retain the primary advantages of the HOPLA, a balance must be struck between sufficient height to exploit such advantage while avoiding intractable complexity.

1.3.1 Analysis

The height problem is the basic constraint upon the capacity of the HOPLA to perform arbitrary operations in a single cycle; but, due to several factors, it does not limit the general purpose nature of the unit. First of all, the range of logical functions that may be deemed to be useful, such as arithmetic functions, are much simpler logically than the class of pathological functions for the same height. Secondly, the major contributor to the exponential increase in height within functions of interest are intermediate values, such as carry bits, which are either best handled by feedback or by recasting the function in a fashion that avoids carries[18]. Finally, the height of the HOPLA is prodigious in comparison to nearly any other scheme using the control operator method. Thus even with a limited number of inputs, the HOPLA should be capable of performing useful functions that exploit its unique features.

Before considering techniques by which the generality of the HOPLA may be extended, it is instructive to examine the extreme limits of width versus height.

If we consider a likely height of the order of 10^6 and if we wish a height:width ratio of $2^n:1$, where n is the number of inputs, then we find that 20 inputs would be maximum. In the case of 20 inputs it is possible to generate all permutations containing all inputs or their compliments and we may then obtain any desired function of the inputs by selecting the appropriate combination of these terms. But if we examine the terms produced, we discover that only one of these terms may be true at one time, thus reducing the P matrix multiplication to a simple though very expensive address decoder. The second stage then reduces to an equally simple yet expensive table look-up system. In the case of greater input variables multiple terms may be true at the same time, and the task of constructing the S matrix becomes more complicated than the enumeration of truth tables. Thus we find that in the case of an "optimal" number of inputs, the system becomes trivial and a greater width is necessary in order to achieve any useful efficiency.

In the same fashion, if we consider a number of inputs comparable to the height and terms in which the maximum fan in is considered i.e. all pixels or their compliments are represented. Then we obtain a set of functions which returns results that answer questions such as "are all inputs true/false","is there one and only one true/false term", et cetera. Such functions tend to indicate global conditions and rapidly increase in height with the complexity of forms. Many of the useful forms of these equations conform to the symmetric functions which are not simplified in electronics due to the existence of a simple means of implementation via cascaded gates[15]. The use of considerably simpler interconnections of gates may be used to generate these functions, reducing the number of intermediate values from 10^6 to ~40. Thus we see that the blind implementation of functions operating over a large number of inputs results in situations as absurd as the case of limited inputs.

The situation for the middle range between these extreme values remains uncertain since at this point the parameters of the HOPLA radically diverge from known electronic or optical devices. Many of the techniques used for logical simplification, such as the Quine-McCluskey algorithm or Spectral Analysis[19,20,21], break down at the large number of variables and functions possible. The only guide to the actual behavior of the functions available at this time is the general properties of logical functions under group theory. From such general results of group theory we find that a single function may be transformed to a wide range of different functions without changing the structure of function by instead changing the nature of the inputs or outputs[19]. The forms of modification that may be used to achieve such transformation are: negation, permutation, replacement with a constant (1 or 0) value, or "modulation" by XORing with another variable which is also input[19,20]. Such operations may be referred to as linear transformations due to their relationship to modulo-2 linear algebra[21]. In a dual rail system this range of operations breaks down to a single basic operation: conditional permutation with the possibility of the condition resting upon a logical constant. This operation is equivalent to the operation performed by the Fredkin gate[22]. A combination of pre-processors and post-processors as indicated in figure 1.7 preforming conditional permutations of the input and feedback variables may well be able to compensate for both the limitations imposed by height and fixed interconnectivity[20].

We may express the linear transformations of the pre- and post-processor in the form of a matrix operation in modulo-2 (mod-2) arithmetic[21]. The practical difference between the mod-2 matrix operations and the boolean matrix operations defined above is that the addition operation in the mod-2 arithmetic corresponds to the exclusive-or (XOR) operation rather than the conventional OR operation. We may express the pre- and post-processor operations in the form of one non-singular (invertible) mod-2 matrix operating upon a vector of the relevant variables (but not their compliments), and a subsequent XOR of the resultant vector by an arbitrary vector[20,21]. The latter operation is necessary to achieve fixed negations and the inclusion of constant values. Karpovsky[21] proves that the overall complexity of such operations is of the order of $n^2/\log_2 n$ for n inputs thus ensuring that the HOPLA is still achieving the bulk of the actual computation. Of course the presence of multiple interrelated outputs complicates the process by the need for insuring that the results are simultaneously post-processed. Thankfully this is possible to achieve by the use of mod-p linear algebras where $p=2^n$.

1.4.Conclusion

The Optical Programmable Logic Array (HOPLA) is a new architecture which uses the N^4 interconnect in concert with thresholding optical inverters to implement the control operator method. The capacity for total interconnection between two two-dimensional planes affords several unique advantages in implementation of the control operator method in comparison with previous architectures. The most notable of all advantages of this architecture is its capacity to operate as a Wave Particle Duality (WPD) processor. The WPD capability of this design suggests that properly

formulated operations may be capable of an energy cost per logical operation which is lower than the limit of kT, a limit independently proposed by both Szilard[23] and Brillouin[24], and first addressed by Landauer[25]. In addition an extraordinary level of parallelism is possible due to the capacity of all inputs to interact in the same cycle.

In exchange for the above advantages the HOPLA has several disadvantages related to the limited "height" of the system and the fixed nature of the interconnects. The extent to which these failings will impact the general purpose operation of the system is uncertain. The above limitations may be ameilorated by the use of pre- and post- processing, careful design, and the use of the "stored program" concept universal to electronic computers.

Chapter 2

Lewin's Algortithm and its Variants

2.1. Introduction

In this chapter two variations upon Lewin's "dual-sense" associative search algorithm[26,27] are proposed. The purpose of these algorithms is the ordered retrieval of a set of logical flags from a large array. The proposed algorithms offer greater performance over Lewin's algorithm in the case of a large number of activated flags. They achieve this by recognizing blocks of flags rather than individual flags. Such a scheme amounts to a compression of the list of flags in a manner that allows for simple decompression. This feature leads to the application of the proposed algorithms to image compression and decompression. It is shown that the proposed algorithms resemble quadtree or octree algorithms in their compression strategies. It is shown that optimal compressions are NP-hard under "worst case" conditions, but arguments are made that specific heuristic functions may be used to obtain high, though not optimal, compression levels.

The two proposed modifications, as well as Lewin's original algorithm, are shown to be well suited for implementation upon certain digital optical computers. A specific architecture for the implementation of the proposed algorithms is described.

2.2. Background

2.2.1. The "Dual-Sense" or Multiread Function.

2.2.1.1. Introduction.

The "dual-sense" architecture was first proposed by Lewin[26] for the ordered retreval of lists from associative memory. The algorithm is equally useful for the ordered retrieval of the addresses of activated logical flags within associative memory. The search method itself is a simple binary search augmented by the "dual-sense" information retrieved. The "dual-sense" or "multiread" function returns clustering information which allows for the efficient pruning of the search tree which leads to greater efficiencies.

The multiread function essentially scans a set of binary numbers for bit positions with the same value in all numbers within the list and returns an expression denoting such positions, which we shall refer to as constraint bits. The resulting expression, which we shall refer to as the constraint bit notation, takes a form identical to that used in logical simplification procedures, i.e. 0 and 1 denoting constraint bit notations, whereas $\emptyset$ denotes variable bit positions where both values of 0 and 1 exist within members of the set.

For example applying the multiread function to the set of numbers (1000, 1010, 1100) results in the expression $1\emptyset\emptyset0$ indicating that the first and last bits are constrained to values of 1 and 0 respectively, while the middle bits contain values of both 1 and 0. It is easily seen from this example that the multiread function is not a one-to-one function since the same result would be obtained from the sets (1000,1010,1100,1110), (1000,1110) or (1100,1010).

Although it is clearly impossible to invert the multiread function, by replacing the $\emptyset$ values with all permutations of 1 and 0 it is possible to construct the largest set which results in a given constraint bit expression, which we shall call the "window set". In the previous example the window set corresponding to the expression $1\emptyset\emptyset0$ is (1000,1010,1100,1110) which shall be denoted by $(1\emptyset\emptyset0)$. All sets which generate the same result are subsets of the result window, although the converse is not true (i.e. if we replace a $\emptyset$ with a 1 or a 0 the resulting window set is a proper subset of the original window set, yet the generating expression is different). The "size" of the window set is defined to be the number of elements within the set and is equal to 2^{f_1} where f_1 is the number of "$\emptyset$"s within the multiread output expression. We may define the multiread function in terms of the window set by stating that the result of the multiread function generates the smallest window set of which the input set is a subset.

The multiread function operates upon sets and cannot distinguish between multiple occurrences of the same value within a list. For this reason the function is particularly well suited to operations involving addresses since the numerical value of two different addresses is, by definition, different. We shall thus confine our current discussion to the use of the multiread operation with addresses. Specifically we wish to map the vector of bits p to a set of addresses P and then to the corresponding

multiread output expression $w(\underline{p})$, in addition we wish to generate the window function corresponding to a given output expression.

We shall describe a means of generating both the multiread function and the window function in both an iterative manner and by the use of boolean matrix functions.

2.2.1.2. Implementation of the Multiread Operation.

We have defined the multiread function in the previous section but we have not given any means by which it may be implemented. Within this section we show the means of implementing the multiread function and the window set function in an iterative fashion.

Given that the constraint bit expression that constitutes the result of the multiread operation is a trinary expression, in order to implement the function in binary logic it is necessary to subdivide the task of generating the result into to parts. Thus we divide the result into two binary vectors $\underline{w}^0$ and $\underline{w}^1$ which indicate by the presence of true (1) value a constraint bit value of "0" and "1" respectively. For example we would indicate the value of $w=1\emptyset\emptyset0$ by the values $\underline{w}^0=0001$ and $\underline{w}^1=1000$.

The task of computing the value of $\underline{w}^1$ then becomes quite simple. The value of $\underline{w}^1$ is simply the bitwise AND of all the elements of the set under consideration. This works since unless all values in a given bit position are true (1) values the result for that bit position is a false (0) value. Following our example, doing a bitwise AND of the values {1000,1010,1100,1110} gives us the value 1000 which is the desired value of $\underline{w}^1$. The value of $\underline{w}^0$ is found by first taking the bitwise logical complement of the elements of the input set and then doing the bitwise AND of the set. This operation simply interchanges the 0 and 1 values and repeats the procedure used to obtain $\underline{w}^1$. Thus taking the logical complements of our example set {1000,1010,1100,1110} results in the set {0111,0101,0011,0001}. When we do the bitwise AND of this latter set we obtain the value 0001 which corresponds to the predicted value of $\underline{w}^0$.

By invoking DeMorgan's law (i.e. a AND b=NOT[(NOT a) OR (NOT b)]=(NOT a) NOR (NOT b)) it is possible to reformulate the above expressions in terms of bitwise NOR (NOT OR) operations. We may thus state that the value of $\underline{w}^0$ is the bitwise NOR of the elements of the input set and $\underline{w}^1$ is the bitwise NOR of the logical complements of the elements of the input set.

2.2.1.3 The Window Set Function.

Just as we have defined the defined the multiread function we have previously defined the window set function $w(\underline{p})$ without explicitly providing a means of generating it. Within this section we describe a serial algorithm for the generation of the window set function, in later sections we shall provide globally parallel algorithms for its generation. Given the generality and utility of this function, simply parallel algorithms have been devised for its implementation upon S-SEED architectures by Murdocca [28].

To iteratively generate the window set of a given constraint bit expression w we first simply replace all $\emptyset$ values with the value of 0 and assign this value to a register $\underline{b}$. We then increment b by one, but we do not pass carries to constrained bits, but rather to the next variable (i.e."$\emptyset$") bit. We repeat this process until all variable bits cycle to a value of 0.

The above algorithm may be implemented quite simply by the use of bitwise AND and OR operations and a increment by one operation. The technique is quite simple: we replace all the constrained bits within b with the value 1. We then increment b by one. Thus any carries that reach a constrained bit carry to the next position until they reach a variable bit. the values of the constrained bits are then returned to their original values.

The explicit implementation follows: The mask selecting constrained bits consists of the quantity ($\underline{w}^0$ AND $\underline{w}^1$). We first set all constrained bits to 1:

$$(\underline{w}^0 \text{ AND } \underline{w}^1) \text{ OR } b$$

We then increment the value by one:

$$(\underline{w}^0 \ \text{AND} \ \underline{w}^1) \ \text{OR} \ b + 1$$

We then set all constrained bits to 0:

$$((\underline{w}^0 \ \text{AND} \ \underline{w}^1) \ \text{OR} \ b + 1) \ \text{AND} \ (\underline{w}^0 \ \text{NAND} \ \underline{w}^1)$$

Finally we reset the constrained bits to their constant values $\underline{w}^1$ and assign the new value to the b register:

$$\underline{b}:=(((\underline{w}^0 \ \text{AND} \ \underline{w}^1) \ \text{OR} \ b + 1) \ \text{AND} \ (\underline{w}^0 \ \text{NAND} \ \underline{w}^1))$$
$$\text{OR} \ \underline{w}^1$$

We begin with $\underline{b}:=\underline{w}^1$ and end when b returns to the value of $\underline{w}^1$. It is a trivial matter to implement this algorithm via either hardwired logic or the use of a microprocessor.

2.2.1.4. Matrix Implementations of the Multiread and Window Functions

The implementation of the multiread operation by means of the boolean matrix operations described in chapter one is now given. We first introduce the $2^m \times m$ boolean matrix S with elements such that:

$$j = \sum_{i=0}^{m-1} 2^i S_{i,j}$$

i.e.:the jth column consists of the value of j in binary. For example the value of S in the case of m=8 consists of:

$$S = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Given an appropriately sized S matrix we may now find the value of the multiread function for a given 2^m long p vector by simple boolean matrix vector multiplication. We express the result in terms of the two binary result vectors $\underline{w}^0$ and $\underline{w}^1$ which are calculated as follows:

$$\underline{w}^0 = [S \ \underline{p}]'$$
$$\underline{w}^1 = [S' \ \underline{p}]'$$

The two m long boolean vectors $\underline{w}^0$ and $\underline{w}^1$ encode the three possible values (0, 1, Ø) of the jth bit position of the output expression in the following fashion: a true (1) value in w^0_j corresponds to a value of "0" in the j_{th} position, a true (1) value in $w1_j$ corresponds to a value of "1" in the jth position, while false values for both the former and the latter correspond to the value of "Ø" in the jth position. If all elements of $\underline{w}^0$ and $\underline{w}^1$ have the value 1 then the window set is the null set.

The generation of the "worst case" window function $\underline{q}(w)$ ($\underline{q}(w)$ AND $\underline{p}=p$ for all p resulting in a multiread value of w) from the value of $\underline{w}^0$ and $\underline{w}^1$ is equally simple. The ith column of the matrix S designates with a 1 all positions of the vector p with a value of 1 in the jth while the complement of that row designates the all positions with a value of 0 within the jth bit position. Thus if we form the intersection of all sets which fulfill the constraints of each individual constrained bit we obtain the window set $\underline{q}$. The intersection of a pair of sets designated by binary vectors indicating membership is simply the bitwise OR of the two vectors. Thus the value of q is simply:

$$\underline{q}(w) = [(S^T \ \underline{w}^0) \ \text{OR} \ (S'^T \ \underline{w}^1)]'$$

Where ^{T} denotes the transpose of the matrix.

2.2.1.5. Properties of The Multiread Function.

Having defined the multiread function and described its implementation, we now derive several useful properties of the multiread function and its corresponding window sets.

We first define the vectors 0 and 1 which are two m long vectors with elements that all contain values of 0 or 1 respectively. In addition we denote the hamming distance between two vectors a and b by dist(a,b). The hamming distance between two vectors is defined to be the number of true bits resulting from the bitwise XOR of the vectors.

Several properties of window sets may be shown by simple enumeration: The size of a window set $q(w)$ generated by a given constraint bit expression w is 2^f, where f^1 is the number of variable (i.e. "∅") bits within the expression. If we replace a variable bit within a constraint bit expression with a constrained bit we obtain an expression with a window set which is half the size of the original window set and a proper subset (i.e. (1∅00) is a proper subset of (1∅∅0)) . If two constraint bit expressions have a bit position in which both are constrained, but to a differing value, the two corresponding window sets are disjoint sets (i.e. (0∅∅0) and (1∅∅0) are disjoint sets).

We say that a window set is continuous if the set of values that it describes are successive. Thus 11∅∅ is continuous since it describes the set (1100,1101,1110,1111) while 1∅∅0 is not since it describes the set (1000,1010,1100,1110) which is not successive. The criterion for window sets which are continuous are that all the variable bits within the expression for the window set be less significant than the constraint bits. Thus 1010∅∅ or 11∅∅∅∅ are continuous while ∅∅1100, 1∅1∅00 and 1'1∅0∅∅ are not. A window set which is not contiguous may be expanded into one that is by replacing every occurrence of a constraint bit which is less significant than the most significant variable bit with a variable bit. For example the discontinuous window set 10∅1∅000 would be expanded to the continuous window set 10∅∅∅∅∅∅.

The intersection of two window sets results in a window set. The constraint bit expression corresponding to the intersection of two constraint bit expressions is obtained by combining the two expressions bit position by bit position, eliminating variable bits in preference for constrained bits (i.e. for a given bit position, if one expression has a constrained value (0 or 1) and the other expression has a ∅ in the same position, then the intersection expression has the constrained bit in that position). For example, the intersection of (1∅∅∅) and (∅∅∅0) is (1∅∅0). Of course, if the two window sets are disjoint, the result is the null set.

The difference of two window sets may result in a null set (if the two window sets are disjoint), or a set of one or more window sets: the difference set. We shall make considerable use of the difference operation in our extensions upon Lewin's Algorithm.

We now present an algorithm to enumerate the set of window sets resulting from the difference of two window sets a-b:

1. Let c=(a∩b). As we have shown in the previous section, c is a window set.

2. We first recognize that a-b=a-(a∩b)=a-c thus reducing the problem to finding the difference between a window set a subset window set.

3. If a=a∩b then the difference a-b is obviously the null set.

4. If c is the null set, then a=a-c=a-b. Thus the result is the window set a.

Otherwise, the window set c is a proper subset of the window set a. Thus the expression for a differs from the expression for c in that c has at least one constraint bit which is a variable bit within a.

5. Let d=a.

6. Find the most significant variable bit within d which is a constraint bit within c. Set this bit to the complement of the value of the constraint bit within c, this expression is an element of the set of window sets forming (a-b). Now set the value of the same variable bit to the value of the constraint bit in c.

7. If d=c then halt, otherwise repeat step 6.

We now present an example:

a=1ØØØ, b=ØØ01
c=a∩b=1Ø01
d=a

d	c	Set Element
1ØØØ	1Ø01	1Ø1Ø
1ØØØ	1Ø01	1ØØØ
1Ø01	1Ø01	Halt

Thus
a-b=1ØØØ-ØØ01={1Ø1Ø, 1ØØØ}

This algorithm operates by recursively splitting the window set d into two halves each window sets, one of the halves being a superset of c and the other a member of the difference set. It is trivial to show that all window sets of the difference set generated do not intersect with each other or with c and that the sum of the sizes of the elements is equal to the difference of the size of a and the size of c.

Given the intersection operation we may compose all window sets from the 2^{m+1} primitive window sets which have only one constraint bit (i.e. 0ØØØ, 1ØØØ, ØØØØ, Ø1ØØ, ..., ØØØ1). We may consider the $\underline{w}^0$ and $\underline{w}^1$ vectors as designating the set of primitive window sets that define the desired window by their intersection. We may use the bitwise OR upon the $\underline{w}^0$ and $\underline{w}^1$ vectors of two different window sets to obtain the intersection of the two sets. By using the intersection operation and the size function we may obtain the necessary information to determine the nature of relationship between two window sets.

We may use DeMorgans law for sets, i.e.: the complementary set of the intersection of two sets is the union of the complements of the two sets, to understand how the multiread function operates. We obtain the set of primitive sets which do not intersect the set P of addresses. The union of these sets corresponds to the complement of the window set we wish to obtain. Thus we obtain the window set by intersecting the complements of the sets described above. In section 2.4 we shall show how this principle relates to simple geometrical principles.

2.2.2. Previous Search Algorithms.

We now describe several methods by which we may search a binary vector p and return an ordered list of values corresponding to the addresses of the true values.

The first algorithm is the simple serial search. This algorithm simply examines each consecutive element of the vector and returns the address if the value of the element is true. For a vector which is $n=2^m$ elements long, the serial search requires n searches to exhaustively search the vector. This search algorithm only requires the value of the element under consideration during each cycle.

2.2.2.1. Frei and Goldberg's Algorithm

The second algorithm is the simple binary search. It requires a function which returns whether the vector contains any true values. The algorithm divides the p vector into two halves. If the first half contains any true values, the process is repeated, recursively dividing the half into quarters and checking their contents. The recursion is terminated when the vector under consideration consists of a single element. If the element is true the address is returned. Once the first half has been searched, the procedure is repeated for the second half. Since during any cycle the algorithm may eliminate half of the remaining search space, under proper conditions this algorithm is considerably more efficient than the serial search.

A specific form of this algorithm was proposed by Frei and Goldberg [29] using the same notation we have used for the description of window sets. Using our notation, the algorithm may be described as follows:

1. Initialize all positions within the window expression w to Ø values.

2. Generate the q(w) vector and form the "dot product" of the q vector and the p vector $f^1=q^T p$.

3. If the value of f^1 is true (1) then find the most significant bit within w which is still a $\emptyset$ value and replace it with a 1. (For example 000$\emptyset$ becomes 001$\emptyset$.) Return to step 2.

If no $\emptyset$ value remains within w then the value of w is a valid binary address of an activated flag. Append the value of w to the list of addresses of true values. reset f^1 to false and continue to step 4.

4. If f^1 is false locate the least significant position within w in which a 1 is present and replace it with a zero. Replace all less significant positions with $\emptyset$ values. (For example 1100 becomes 10$\emptyset\emptyset$.). If there are no positions within w which have the value of 1 then halt, the search is completed.

The case of (1000,1100,1010) is given as an example

w	f^1	Comments
000$\emptyset$	1	
100$\emptyset$	1	
110$\emptyset$	1	
111$\emptyset$	0	
110$\emptyset$	1	
1101	0	
1100	1	Add to list
100$\emptyset$	1	
101$\emptyset$	1	
1011	0	
1010	1	Add to list
100$\emptyset$	1	
1001	0	
1000	1	Add to list
000$\emptyset$	0	Halt

Frei and Goldberg's algorithm is clearly a binary depth first search. The value of f^1 indicates the presence of one or more true values within the window prescribed by w. Each time a 1 value replaces a $\emptyset$ value the search window is cut in half. Each time a 0 value replaces a 1 value the search window abandons the current branch and tries the next possible branch. Each time step 4 is invoked the result is equivalent to a recursion in conventional binary searches. Once the final branch is exhausted the routine halts.

Frei and Goldberg's algorithm forms the basis for all algorithms described within this chapter and displays several common features. It implements a recursive algorithm in a serial fashion, eliminating the traditional complications associated with the implementation of recursive algorithms. It requires the capacity to select a specific search window and extract global information on the selected window.

The primary weakness of Frei and Goldberg's algorithm is that it must examine branches that contain no true values before abandoning them. This requirement arises since this is the only means by which the algorithm may gain information on the distribution of true values. By introducing a more sophisticated heuristic function than f^1 Lewin's algorithm eliminates such false starts.

2.2.2.2. Lewin's algorithm

The Lewin algorithm achieves a greater efficiency over Frei and Goldberg's algorithm by using the more sophisticated (and complex) multiread function to eliminate wasteful searches. The difference between the two algorithms is that the Lewin algorithm uses the multiread function to determine the smallest search window that encloses the remaining true values, and then replaces the current search window with this value. This achieves two functions: it reduces the search window and it ensures that when the search window is split both subwindows contain true values. The latter feature arises from the fact that when a search window is split in half the resulting subwindows are windows as well. Since the original window is the smallest window that encloses all true values, both of the subwindows must have some true values enclosed.

In our notation the Lewin search algorithm may be expressed as follows:

1.Initialize two search window expressions r and w by setting all positions to a value of Ø.

2.Generate the window set $\underline{q}(r)$ and do a bitwise AND of q with $\underline{p}$.

$v = \underline{q}(r) \text{ AND } \underline{p}$

3. Do a multiread of the vector v to obtain the window expression w from the values of w^0 and $\underline{w}^1$

$$\underline{w}^0 = [S \ \underline{y}]'$$
$$\underline{w}^1 = [S' \ \underline{y}]'$$

4. Find the most significant position within w that contains a Ø value. Replace the same position within r with a 1. Return to step 2.

5. If there are no remaining Ø values within w then w is a valid binary address of a true value. Append w to the list of true values and continue to step 6.

6. Find the least significant position within r containing a 1 and replace it with a 0 and set all lower significance positions with Ø values. Return to step 2.

6. If there are no positions within r that contain a 1 then all true values have been found, halt.

An example of the Lewin Algorithm is now given for the case of (1000, 1100,1010):

r	w	Comments
ØØØØ	1ØØØ	
Ø1ØØ	1100	Add to list
ØØØØ	1ØØØ	
ØØ1Ø	1010	Add to list
ØØØØ	1000	Add to list,Halt

As we may see in the example given, the Lewin algorithm is considerably more efficient than Frei and Goldberg's algorithm. Surprisingly, the number of cycles required to complete an ordered search is not a function of the size of the search space but of the number of true values[26,27]. If there are h true values 2h-1 cycles are required in order to retrieve the addresses of all true values.

2.3. The Improved Lewin Algorithms.

2.3.1. Introduction.

The Lewin algorithm displays impressive efficiency in searching sparsely populated search spaces, but loses efficiency as the population of the search space becomes uniformly distributed. When the true values within the search space are uniformly distributed, the smallest window that encloses all true values is often identical to that of the search space itself. Under such circumstances, the multiread function returns no useful information, at which point the Lewin algorithm becomes functionally identical with Frei and Goldberg's algorithm.

The worst case condition for Lewin algorithm occurs, paradoxically, when the entire search space is populated with true values. Under such conditions, the Lewin algorithm requires 2n-1 cycles to search the space while a serial search only requires n cycles.

An especially odd feature of the above paradox may be illustrated by the following example. Consider a Lewin search over a set of addresses (ØØØØ) in which all eight odd values (ØØØ1) are true. In the first cycle of the Lewin search the multiread function returns a value for w of ØØØ1, precisely the set of true values! Unfortunately sixteen more cycles must be wasted to enumerate all of the true values. The reason for this is that a multiread function result of ØØØ1 may be the result of true value sets that may range between (0001,1111) and (ØØØ1). Without additional information it is necessary to proceed with additional, possibly wasteful, cycles to finish the search.

2.3.2. The First Algorithm.

The simplest method by which the deficency noted may be addressed is to recognize when the search window is devoid of false values and, if so simply enumerate the entire window in the manner of

section 2.2.1.2. rather than search it. We introduce the function $f^0 = \underline{q}(w)^T \underline{p}'$ which is identical to the function f^1 defined previously, except that it indicates the presence of false values within the selected window rather than true values.

We may then simply append an additional step to our formulation of the Lewin Algorithm in section 2.2.2.2.:

2.a. Let $f^0 = \underline{q}(w)^T \underline{p}'$. If f^0 is false then append w to the list of true values and proceed to step 6.

By expressing the list of true values in terms of window sets, rather than individual addresses, this algorithm not only achieves an increase in the search rate, but also a compression of the list of true values. As we shall see, the capacity of this algorithm for compression may prove quite useful in its own right.

Like Frei and Goldberg's algorithm, the present algorithm first examines a given branch, and then "discards" it. The difference between the two algorithms is that the former discards empty windows while the latter records and then discards full windows. The weakness of the two algorithms are the same, insufficient information leads to unnecessary searches. Following this analogy to its logical conclusion suggests that the use of the multiread function upon the complement of the search space may serve to correct this problem. This is the basis of the second proposed algorithm.

2.3.3. The Second Algorithm

The second proposed algorithm addresses the deficiencies of the first algorithm by applying the multiread function to both the search space and its complement. We designate the result of the multiread operation as applied to the search space as the true window and denote the returned expression as l. In the same fashion we designate the result of the multiread operation as applied upon the complement of the search space as the false window and denote the returned expression as $\underline{d}$. The associated logical operations are:

$$l^0 = [S \; \underline{q}(w)^T \underline{p}]'$$
$$l^1 = [S' \; \underline{q}(w)^T \underline{p}]'$$
$$d^0 = [S \; \underline{q}(w)^T \underline{p}']'$$
$$d^1 = [S' \; \underline{q}(w)^T \underline{p}']'$$

Initially, one might assume that the information returned by the l and d functions might introduce considerable complications given the range of possible relationships between the two functions. Surprisingly, there are only three possible outcomes in terms of the set relationships between l and d. Either both l and d are both the same set as the search space r, or one (but not both) expressions are proper subsets of the search space if not a null set, while the other is the same as the search space. The third option is that l and d are disjoint window sets with a union that forms the window set that describes the search space.

This situation arises due to the fact that the size of all window sets are powers of two. It follows that all windows enclosed within a window must be at least half the size of that window. Thus if l is a proper subset of the search space ·, then l must be at least half the size of r. Since l encloses all true values within it, the number of true values within the search space must be equal to or less than half the size of the search space. If precisely half of the search space consists of true values then every element within l is true, since l is half the size of the search space and encloses all true values. In this case, the window set d which encloses all false values is also a subset of the search space, one which is disjoint from the set l. If less than half of the search space consists of true values then it follows that more than half of the search space consists of false values. Since the window set d must contain all false values within the search space its size must the greater than half of the search space. But, as we have noted, no subwindow of the search space may be larger than half the size of the search space. Thus in this case d is the same set as the search space r. This proves the contention of the previous paragraph.

Each of the predicted outcomes may be used within our algorithm to extract information relevant to the position of true values. We shall now enumerate the possible outcomes and discuss the information that may be extracted.

The first possible outcome is that both window sets l and d are identical to the search space r. In this case, the true and false values within r are are so interspersed that it is necessary to recursively subdivide the search space to obtain further information.

The second possible outcome is that either l or d return the null set. In this case the entire search space is either completely filled with false or true values respectively. In either case the disposition of the entire search space is known.

The third possible outcome is that l is a subwindow of the search space r. This is identical to the collapse of the r window in the previous algorithms. It indicates that the section of the search space external to the l window set consists entirely of false values and may be discarded. The search space r is set equal to l and the procedure is repeated. Note that this contingency also addresses the case where both l and d are subwindows.

The fourth possible outcome is that d is a subwindow of the search space r. This outcome is the most interesting, in that this is the outcome which is unique to the present algorithm. It indicates that the section of the search space external to the d window consists entirely of true values. Obviously, this external area contains no further information and may be discarded once its extent is noted. Unfortunately, the extent of the external area is the difference between two window sets, an entity which in general does not constitute another window. Thankfully as we have shown in section 2.2.1.5. it is possible to express the difference between two windows as the union of several window sets, although it is preferable to express the area in terms of the difference so as to retain the highest possible compression. Thus the proper response to this outcome is to record the difference of r and d and then set r equal to d and repeat the procedure.

Unfortunately, the use of set differences to describe blocks of true values may result in the loss of order, since the contents of the remaining search space may proceed the contents of the window set difference. If ordered retrieval, as opposed to pure compression is required, it is possible to limit the collapse of the search space so that it is contiguous, and divide the difference set into windows that precede and follow the search space. The preceding element may then be added to the list, and the following section pushed upon a stack, which is then popped after the termination of the current recursion. Such a modification results in a reduction in performance since the extent of collapse is limited.

All of the possible responses to the outcomes of the l and d functions result in the reduction or "collapse" of the search space to the smaller of the resulting window sets regardless of whether the eliminated section is all true or all false. Two important conclusions result from this observation. The first is that this algorithm does not search for blocks of true values as do the other algorithms, but instead, searches for blocks of interspersed true and false values within a backround of undifferentiated values. In essence this algorithm detects boundaries rather than blocks. The second conclusion is that a of series of collapses may occur over a continuous period of operation without recursive splitting of the search space as long as the backrounds alternate in value.

Having analyzed the implications of each possible outcome of the application of the multiread function to both the search space and its complement, it is now possible to explicitly formulate our second algorithm. We shall use two window sets r and v the former describing the search space, while the latter is used to track the progress of recursion. It is important to note that r is always a subset of v.

1. Initialize both r and v to include the entire search space.

2. Determine the values of l and d for the current value of r.

3. If l is a null set then proceed to step 9. Record a termination of recursion.

4. If d is a null set then proceed to step 9. Append l to the list of true values. Record a termination of recursion.

5. If l and d are disjoint sets then proceed to step 9. Append l to the list of true values. Record a termination of recursion.

6. If d is a subset of r then set the value of r equal to the value of d. Append the set difference of l and d to

the list of true values. Proceed to step 2 unless d contains no Øs. If d contains no Øs proceed to step 9 and record a termination of recursion.

7. If l is a subset of r then set the value of r equal to the value of l and proceed to step 2 unless l contains no Øs. If l contains no Øs, append the value of l to the list of true values and proceed to step 9 and record a termination of recursion.

8. Find the most significant position within r with a value of Ø and replace it with a 1. Replace the same position within v with a 1 Proceed to step 2.

9. Find the least significant position within v containing a value of 1 and replace it with a 0. Set the value of r equal to that of v. proceed to step 2. If there are no remaining values of 1 within v Halt.

Analyzing the above algorithm, we see that 3-7 are responses to the various results of l and d discussed in the previous paragraphs while steps 8-9 mediate the recursive splitting of the search space. Steps 8 and 9 are extensions of the corresponding steps 4 and 6, respectively, within the Lewin algorithm. The important difference is the presence of the window set v which keeps track of of the reduction of the search space due to recursive splitting alone. The need for v arises due to repeated collapses of the search space. If we were to operate upon the search space r rather than v we would find that we would recurse into areas that had already been eliminated by collapses. In essence v operates like a stack to keep track of recursive splitting.

Applying our current algorithm to our ongoing example {1000,1100,1010} we obtain the following results:

w	l	ā	Comments
0000	1000	0000	
1000	1000	1110	Add 1000-1110 to the list and Halt.

In this example the second algorithm is much more efficient in both speed and compression than the previous algorithms. In addition, no recursive splitting is required, and two sequential collapses occur, first a collapse of the l function and the second a collapse of the d function. Sufficient information is supplied to select the most efficient separation of the search space into subwindows.

A more complete example {0000,1000,1010,1110,0111} illustrates more of the features of the algorithm:

v	w	l	d	Comments
0000	0000	0000	0000	
1000	1000	1000	1000	Split
1000	1000	1000	1110	Add 1000-1100 to list. End recursion
0000	0000	0000	0000	
0100	0100	0111	0100	Split Add 0111 to list End recursion
0000	0000	0000	0000	Add 0000 to list End Recursion Halt.

This example displays all of the critical elements of our second algorithm, the use of both the l and d functions to achieve collapses, the use of the v and r window sets to achieve implicit recursion without the use of a stack.

Like the first algorithm we have proposed, this algorithm achieves improved performance by recognizing blocks of constant value, and in doing so not only achieves ordered retrieval but also serves as a compression algorithm. Ultimately, the significance of the second algorithm is as a compression algorithm.

2.3.4. Extensions Upon the Second Algorithm

As we have seen, our second algorithm improves upon Lewin's algorithm by the use of more sophisticated versions of the multiread function. There are several variants upon the second algorithm which attempt to improve its performance even more by extracting further information. We

shall see that optimal compression is achieved by an algorithm which is NP-hard, but that less sophisticated algorithms may result in improvements in special cases.

2.3.4.1. Selective Splitting

Let us consider a set of addresses (ØØØ1,ØØ10). Although this set consists of only two window sets, our second algorithm shows little improvement in preformanceover Lewin's original algorithm. In comparison, the set (01ØØ,10ØØ), is decomposed into the two window functions by the second algorithm in three steps, one split and two collapses. Although the two examples are of the same complexity, the performance of the second algorithm varies radically. The reason is that the latter set contains two contiguous window sets, whereas the former contains two non-contiguous window sets which are interspersed. Since our second algorithm is designed to achieve ordered retrieval, it employs recursive splits which divide the search space into contiguous window sets. Algorithms which select the order of the recursive spliting in response to some strategy may achieve improved compression at the cost of abandoning ordered retrieval.

Selective splitting algorithms are of increased complexity since the implementation of the recursive splitting requires the explicit use of a stack.

Unfortunately, the decomposition of a given set of addresses into a minimal combination of window sets is identical to the problem of optimally minimizing a logical function in the sum of products form, a problem which is known to be NP-hard. Thus optimal compression of arbitrary data may only be achieved at the cost of unacceptable degradation of performance. Although it is not possible to always achieve optimal performance, algorithms employing carefully constructed heuristics may approach optimal performance under practical conditions. The most obvious strategy for selecting recursive splits is the use of an exhaustive search of all possible splits that may be employed. Although ensuring optimal results, the exaustive search also results in NP complexity due to the combinationally explosive number of permutations of splits that must be examined.

By limiting the degree to which we look ahead, and then selecting the combination of splits that results in the best results, we may limit the complexity of our search, yet improve upon our compression in some cases. Unfortunately, the improvement in compression performance seldom justifies the considerable increase in overhead resulting from such strategies.

2.4. Applications

We now consider several potential applications for both our algorithms and the multiread function. Besides the obvious application to associative memory, applications exist in the monitoring and control of parallel architectures and network systems, and most importantly image decomposition and analysis. By providing a geometrical interpretation of our algorithms, we illustrate the utility of our second algorithm and its relationship to several well known image compression algorithms. We also present several variants of our second algorithm which implement existing compression strategies in a highly parallel fashion.

2.4.1. Associative Memory

Associative or Content Addressable Memories (CAM) are memory units which select data elements based upon various properties of the data itself, as opposed to conventional Random Access Memories (RAM) which select data elements by their location, as described by an address. CAM memories would have consiserable advantages over conventional RAM memories in many applications, such as database query, memory management, artifical intellegence, genome sequencing, electronic warfare, and, air traffic control[30,31]. Highly sophisicated CAM implementations may be considered parallel computer architectures in their own right. Despite considerable effort[30,31] electronic implementations of CAMs have proven impractical, with the exception of niche applications such as memory management.

Optical implementations of content addressable memories have been the subject of considerable effort[3-5,28,32-35]. Four major lines of assault have been taken: Holograms intrinsically display simple associative behavior which may be coupled with their potentially immense information storage capacity to produce, simple best match or key search associative memories[32]. Optically implemented neural nets also display associative behavior which may be coupled with other properties of neural nets to achieve desired tasks[33,34]. Highly interconnected optical architectures either fixed or dynamic may be used as CAMs (in fact the HOPLA may be interpreted as a fixed

storage CAM)[3-5]. Finally, simply parallel collections of identical processors may be used to implement highly sophisticated associative memories in a manner similar to that of electronic implementations of CAMs[35]. Our algorithms may be used to considerably augment architectures employing this final strategy.

Conventional digital associative memories, or content addressable memories, operate by simultaneously performing various search operations upon a block of memory containing various values. By performing operations simultaneously, it is possible to search large blocks of data for various properties very quickly. Such memories essentially constitute a special purpose parallel architecture for information retrieval of information. The bulk of the operations performed are simple identical logical operations which are performed simultaneously upon every word of data within the memory in either a bit serial or a word serial fashion. Such operations are easily implemented within simply parallel Single Instruction Multiple Data (SIMD) machines, and several optical implementations have been proposed[35,36]. But certain associative memory operations require globally parallel logical functions to operate successfully. It is these global functions which are of current interest.

Certain associative searches, such as the search for the minimum value within a memory, cannot, by definition, result in the elimination of all values within the memory (i.e. there is always a minimum value). Such searches cannot be implemented by purely employing operations which operate upon each word of data independently. The reason for this is that such independent operations could simultaneously eliminate from the search every remaining value under consideration. It is necessary to employ a global function to monitor the simply parallel search function, so as to ensure that at least one value remains under consideration after every parallel search operation. A variety of global monitoring functions exist, each with varying levels of complexity and sophistication. Not surprisingly, the multiread function and its variants which have been described in section 2.2 are of considerable utility in this regard.

In order to illustrate the utility of the multiread function for such associative search functions, we present a practical algorithm for the extraction of the minimum value from a bit serial associative memory. The specific algorithm is an extension of one proposed by Morozov[3] for use on optical computers.

The algorithm is as follows. We progressively examine each bit position from the most significant bit to the least. If the value of a given position is identical for all elements of the list we discover nothing and skip to the next less significant bit. In the case where the bit values differ, the list elements which have a 1 in the position under examination are greater than the with a 0 in the same position and thus are eliminated from the list and the next less significant bit is examined. If at any time there only exists one remaining element to the list this is obviously the minimum and we may halt further examination. If we have more than one remaining element to our list when we have examined the least significant bit the remaining elements are minimum and equal. His algorithm allowed these operations to proceed in parallel using the operator method.

Morozov's algorithm is illustrated in figure 2.1. The masked area is indicated by a bold line. The shaded area indicates the situation where all remaining numbers have a value of 1 at the bit under consideration.

Given a list of 2^m binary numbers m bits long we define $A_{i,j}$ as the ith bit of the jth number. We then construct a table q_j and initialize all elements to 1 (true). We then follow the procedure below:

1. Let i=m

2. For all j=0...n-1 let $p_j = A_{i,j}$ AND q_j.

3. If one and only one value of p_j remains true then the position of the true value denotes the unique minimum. Halt and return the address of the true (1) value.

4. If all values of p_j are false (0) then discard p_j, otherwise let $q_j = p_j$.

5. Decrement i by one. If i=0 then there are multiple positions where the same minimum occurs each designated by $p_j=1$ for that position. Otherwise Repeat the process from step 2.

By examining figure 2.1 we may see the means by which this algorithm works. If we examine the most significant bit of our list of numbers it is obvious that numbers with a 1 in the most significant position are greater than those without, and thus may be excluded from the list of candidates for the minimum. If we proceed to the next most significant bit we may use the same criterion to eliminate candidates. But if all values remaining in our list of candidates have bits of value 1 at the column under consideration our criterion fails and we must skip our elimination at this cycle. Steps 2 and 4 of our procedure implement the elimination and check skip phases of our algorithm. The q_j table serving to indicate membership in the candidate list. Obviously if a single position remains on our candidate list, it is the minimum and we may halt immediately and eliminate useless cycles. Step 3 implements this phase of our algorithm. Finally it is just as obvious that if we cycle through all of the columns of the list and still have multiple positions the values still in the list are equal and minimum. Step 5 of our procedure implements this phase of our algorithm. By varying the operations performed in step 2, a broad range of different searches (maximum, greater than value, less than value, closest hamming distance to value, etc.) may be performed

With the exception of step 3 all of operations within this algorithm are simply parallel and may be implemented upon a range of simply parallel optical architectures. Step 3 on the other hand is intrinsically global in nature. We may implement step 3 via the use of the multiread function. If we apply the multiread function as defined within section 2.2.1.4. to the vector p

$$\underline{w}^0 = [S\ \underline{p}]'$$
$$\underline{w}^1 = [S'\ \underline{p}]'$$

we obtain the window set w. By examining w we may determine whether there are any true values, and if so, if there exists only one true value. We may distinguish these results in the following manner:

1. If w is the null set (i.e. $\underline{w}^0 = \underline{w}^1 = 1$), then there are no true values.

2. If w contains variable bits (∅) (i.e. $\underline{w}^0 \neq [\underline{w}^1]'$) then there exist a multiplicity of true values.

3. If w only contains contains constraint bits (0,1) (i.e. $\underline{w}^0 \neq [\underline{w}^1]'$) then there is only one true value and it is located at the address $\underline{w}^1$.

Thus the multiread function is sufficient to provide the needed global information for this algorithm. Although closely related logical functions may be employed to obtain the same results (see Appendix A), the multiread function has the added advantage of supporting ordered retrieval.

If the associative search algorithm terminates with multiple equal minima, we may employ Lewin's algorithm or our extensions to extract an ordered list of their positions. Such an ordered retrieval provides a rapid method of converting the results generated by the associative search algorithm from a sparse collection of flags distributed across a large store, to a compact sequential list well suited for output.

2.4.2. Parallel Processor and Network Monitoring

Several commonly used architectures for parallel processors have topologies based upon the relationships between the addresses assigned to each processor. Both the multiread function and our proposed algorithms may be employed to extract useful information about processor activity within such architectures.

One common example is the hypercube architecture. This architecture consists of 2^m processors, each processor connected to m other processors with a toppology identical to that of a unit m-cube. Each of the processors has an unique m bit address. Each processor is connected to the m processors with addresses which are a hamming distance of one from the address of the current processor. Such an arrangement has many well known advantages, such as redundancy, simple routing, and a maximum distance between processors of m intervening processors.

A 2^m processor hypercube network may be split into two independent 2^{m-1} processor hypercube sub-networks, and each of these networks may be similarly subdivided. The list of addresses for any of these independent sub-networks may be described as a window set. Thus it is conceivable that the multiread function, and our proposed algorithms may prove useful in the task of processor allocation.

One possible scenario for the use of our methods is as follows. Consider an algorithm for which the task for each processor requires a differing number of cycles to accomplish. Once a processor has finished a task it lays idle. If we construct an 2^m long vector p of flags with a flag corresponding to each processor which is set to true if the processor is active, and false if the processor is idle. If we perform the multiread function upon p we obtain the expression w which describes the smallest sub-network that is still active. By generating the difference between the window set of all processors (all "Ø"s) and w we obtain a set of window sets which ennumerate a partial list of idle sub-networks which may be allocated to other tasks. If we complement p and perform our second algorithm upon it, we obtain a list of all idle sub-networks which may be allocated for other tasks. Although the second algorithm is more comprehensive, the multiread function used alone is faster and consumes a fixed number of cycles.

If necessary, we may accelerate the operation of our second algorithm for this process by terminating recursive descent when size of the the search space r falls below a set value. This would discard sub-networks too small to be useful, including individual processors. Selective splitting algorithms may also prove useful in avoiding the bias toward contiguous window sets within the second algorithm.

Many of the properties illustrated within the above example may be applied to other architectures and network switching systems that use similar interconnection schemes. Schemes which use window sets (under a different name) to broadcast information to multiple processors have been proposed for SEED architectures[28]. Thus it is quite conceivable that our algorithms may find application in multiprocessor and network management.

2.4.3. Image Decomposition and Compression.

Although we have proposed several other applications for our algorithms, it is image decomposition and compression which is the most promising. When the our second algorithm is extended to problems involving multidimensional data, it is seen that its results have intrinsic geometrical meaning. Furthermore, it may be shown that these results are closely related to the well known bintree, quadtree, and, octtree representations of images and volume. Finally we shall show how our methods may be used to arrive at globally parallel implementations of well known quadtree algorithms.

Within this discussion, we shall primarily consider images consisting of black and white bitmaps, that is, images with bits that are either black (false) or white (true). We shall then consider extensions upon our basic algorithms in order to handle grey scale and color images.

2.4.3.1. Two Dimensional Extensions of the Multiread Function.

In order to extend our algorithms to two dimensions, it is necessary to devise an appropriate mapping between a two dimensional bitmap and a one dimensional vector of flags. Various different mappings result in differing geometrical interpretations of the results of the multiread function and the window set function. We shall consider a range of different mappings and the resulting interpretations.

Let us consider $2^{m/2} \times 2^{m/2}$ pixel black and white image (we assume that m is even. This image is stored within the matrix $I_{x,y}$ as a set of boolean values, white being denoted by true (1) and black by false (0). We must then map this image to and from a 2^m boolean vector p in order to process it using our algorithms. We may describe the mapping in terms of the operations applied to the indices of p and I:

$$p_{i(x,y)} = I_{x,y}$$
$$I_{x(i),y(i)} = p_i$$

where i is the index for p and x and y are the indicies for I.

2.4.3.1.1. The Raster Scan.

The most obvious mapping is the raster scan, which maps pixels from left to right and then top to bottom as per figure 2.2. The functional dependence of the index k is $k(x,y) = x + 2^{m/2}y$. Since we have selected an image with dimensions which are a power of 2, the high m/2 bits of k are equal to y and

the low m/2 bits of k are equal to x. For example if x=1010 and y=1100 then k=11001010 (all quantities binary).

The segregation of bits within the raster scan order, leads to a very interesting property; the window sets generated by the multiread function are segregated as well, and we may divide the result of the window function into two separate expressions which govern the constraints imposed upon the x and y indicies separately. Thus if we refer to the upper left hand example in figure 2.3, we see that if we perform the multiread function upon the image shown the result is w=11ØØ00ØØ. If we then superpose the window set prescribed by w (the cross hatched region), we see that the result forms a square block within the image that encloses the true pixels within it. The remaining examples within figure 2.3 indicate various other possible window functions enclosing various pixel configurations. We see that the windows are rectilinear, and if not contiguous, periodic, and the dimensions and periodicity of the windows are powers of two. Even more varied patterns may result in larger images, such as periodic sequences of periodic rectangles, although these seldom occur in practice.

This interpretation allows us to visualize the operation of the multiread function. We may liken the multiread function to an "elastic" frame which when placed within an image, collapses down to the frame with the smallest area that contains all of the true pixels within the image. Of course, the resulting frame is a window set and is thus constrained to have sides with lengths that are a power of two. The utility of the multiread function becomes readily apparent, when it collapses, it eliminates from the search space the empty background sections of the image.

2.4.3.1.2. Space Filling Curves.

The raster scan mapping displays geometrical significance only when the width of the image selected is a power of two. Furthermore, window sets which are contiguous in two dimensions do not retain their continuity when mapped to the p vector. A matter of some concern given the weakness of our algorithms with regard to the detection of discontinuous window sets. It would be much preferable to employ a mapping which avoids these problems. Space filling curves have been employed with considerable success in mapping multidimensional data to one dimension[37-40], their use in image proccessing and compression[40-41] for the same purpose is well known. Within this section, their application to the current algorithm shall be shown.

A space filling curve is a parametric curve with an argument which continuously varies between 0 and 1 and which visits every point within a unit area (or multidimensional volume). For any given value of the argument there exists a coordinate corresponding to that point along the curve, and for any coordinate there exists a corresponding argument for the function. Thus the space filling curve maps a continuous area (or multidimensional volume) to a continuous unit length. In addition the space filling curve exhibits locality, that is, two points which are nearby in the unit area will generally be nearby in the unit length.

Given a discrete grid of values, such as an image consisting of pixels, it is possible to map this matrix to a vector using an approximation of a space filling curve. Algorithms to generate such approximations[37-39] are well known, and are well suited for generation and use by digital computers.

A good example of such curves is the Hilbert curve (figure 2.2) an example of the family of curves known as Peano curves. The Hilbert curve is especially well suited to our use, since it maps a $2^{m/2} \times 2^{m/2}$ grid to a 2^m line. An algorithm which generates the appropriate mapping and its inverse using only simple bitwise operations[37].

Hilbert curves are well suited for use in our algorithms, in that they map contiguous window sets from images to sequential window sets in the p vector. Additional advantages to the use of the Hilbert curve, is that it is well defined and understood, and that it is readily extended to higher dimensions.

An alternative to the use of the Hilbert curve is to interpose the bits associated with the x and y coordinates to generate the value of k(x,y) (i.e. x=1111,y=0000,k(x,y)=01010101). This mapping is known as the Morton order and it is commonly used within image processing applications[42]. Like the Hilbert curve, they map contiguous window sets from images to sequential window sets in the p vector. The advantages of the Morton order are that the mapping algorithm is considerably simpler than that required for the Hilbert curve, and that the orientation of the path is the same for each level of recursion. The multiread and window set function for the Morton order and the Raster Scan Order differ only in the order of the output bits, which are interspersed in the same manner as described above.

Several other close variants of the mappings described above exist, and shall be described when we consider issues of the physical implementation within section 2.6.

2.4.3.2. Algorithms.

2.4.3.2.1. The Second Algorithm in Two Dimensions.

Within section 2.3 of this chapter we have described a formulation of the second algorithm in a rigorous and abstract manner. Within the current section we present a geometrical interpretation of the second algorithm, which shall illustrate the utility of this algorithm for the compression of images.

As we have noted in the previous section, we may view the output of the multiread function as an "elastic" frame which collapses down to the smallest window set that encloses all of the true (light) values. The l and d functions employed within the second algorithm operate in a similar manner. The l function is identical to the multiread function, and thus encloses all of the true (light) pixels. The d function operates upon the complement of the image, and thus encloses all of the false (dark) pixels. The collapse of either l or d eliminates a background region which is respectively all dark or all light. Essentially, by employing both functions we obtain a window which collapses upon the section of the image within which a mixture of light and dark pixels exist. By recursively applying these functions it is possible to collapse this window until the entire image is subdivided into disjoint light and dark window sets or differences of window sets. This forms the basis of our image decomposition algorithms.

Of course, as we have noted in section 2.3, there are six possible outcomes which may occur when the l and d functions are applied, we now illustrate the geometrical interpretation of each outcome within figure 2.4.

In the first case both l and d collapse, indicating that the light and dark regions split into two disjoint window sets (the left and right halves in this case). The window set for the light region is sufficient to describe the entire image and no further steps are necessary.

In the second case only the d function collapses (to the lower right quadrant), implying that the remainder of the image is a light background. We record the light background as the difference of the search space and the window indicated by the d function. We then repeat the process with a search space limited to window indicated by the d function.

In the third case only the l function collapses (to the lower right quadrant), implying that the remainder of the image is a dark background. We discard this background and repeat the process with a search space limited to the window indicated by the l function.

In the fourth case the d function returns a null set, indicating that the entire image is light. We record the search space window set and halt.

In the fifth case the l case returns a null set, indicating that the entire image is dark. We halt.
In the sixth case neither function collapses, the light and dark pixels are sufficiently interspersed so as to not have a background as such. We recursively split this image in half and repeat the procedure with the two halves.

Figure 2.5 illustrates the decomposition of an image in this fashion. In this specific case, the decomposition consists of an alternating series of collapses of the l and d function. As we may see, the search space rapidly contracts as a result of repeated collapses. In addition, all but one pixel effectively consists of "background" as we have defined it. Essentially, the second algorithm does not seek blocks, it seeks backgrounds, or more accurately boundaries.

The strategy we have outlined describes the application of the second algorithm to images with one exception: the order in which we split the image when no collapse occurs. This order is dictated by the mapping between the image and the p vector that we employ.

Figure 2.6 indicates the order of splitting which arises from the raster scan order and the Morton order (the Hilbert curve order is essentially the same as the latter). In the case of the raster scan the splitting occurs along one axis until such splits are impossible and then proceeds along the other axis. The Morton order, on the other hand alternates axes, bisecting the image each time. In the case of images, the latter strategy is clearly preferable. Figure 2.7 indicates two simple images, the Morton order decomposes both with equal ease, whereas the raster scan order is much less efficent when

applied to the second image. This is essentially a geometrical formulation of the argument presented in section 2.4.3.1.2 for the use of space filling curves. Of course, although the use of space filling curves reduces the occurence of interspersed discontinuous window sets it does not eliminate their occurrence. The simple checkerboard in figure 2.8, although consisting of the two window sets shown, results in worst case performance when decomposed by the second algorithm.

We have shown that the second algorithm, properly applied to an image, decomposes it recursively into a series rectilinear blocks, even though the second algorithm was not initially formulated with reference to geometry.

2.4.3.2.2. The Second Algorithm in Higher Dimensions.

The second algorithm is equally well suited to the decomposition of "images" of higher dimensions. An example of such an "image" is the series of cross sections generated by CAT scans. The interpretation of the operation of the algorithm in higher dimensions yields insight into its operation in two dimensions.

Figure 2.9 indicates the decomposition of a periodic image by the second algorithm. The Morton order is employed. As with the case of the decomposition depicted within figure 2.5 the process consists entirely of collapses. The window sets which are employed with figure 2.9 are discontinuous periodic window sets in two dimensions. If we interpret this image as four 4x4 cross sections of the solid indicated, the window sets may be interpreted as contiguous three dimensional window sets. Either interpretation is valid. We may interpret discontiguous window sets as projections of contiguous multidimensional window sets. This interpretation implies that the multiread function will collapse to the smallest window set that will enclose the true values, regardless of the dimension of the window set. The concept of dimension in either case is essentially an arbitrary assumption. We may regard our algorithms as essentially m dimensional and view all window sets as projections of contiguous m dimensional window sets.

Although the multiread function is essentially independent of the dimension employed, the order of splitting is not. The order of splitting employed within the Morton order is dependent upon the explicitly two dimensional nature of the mapping. This results in an order of splitting which does not alternate axes, but rather splits the z axis repeatedly, and then alternates between the x and y axes. This may be corrected by employing a three dimensional extension of the Morton order, which intersperses bits of the x, y, and z indices. As in the case of the original Morton order, the multiread functions of the raster scan order and both the two and three dimensional Morton orders differ only in the order of the output bits.

The above observations do not strictly hold for the Peano (Hilbert curve) order. Although the Hilbert curve is self similar in shape, it is not self similar in orientation. The repeated rotations of the primitive curves that compose the Hilbert curve lead to non-periodic discontinuous window sets. Such a property tends to supress collapses to windows which are discontinuous in the dimension that the Hilbert curve has been generated. We may use the Peano order for the compression of multidimensional images, but we must employ mappings based on Hilbert curves of higher dimensions[37,40].

2.4.3.2.3. Quadtrees

We have shown how we may apply the second algorithm to decompose two and three dimensional images into a series of disjoint window sets. Within this section we shall discuss the application of our results to image processing and compression. We shall show that the representation of the image generated is closely related to the well known quadtree image representation and its variants. Finally, we shall show how we may adapt the broad range of existing quadtree algorithms to highly parallel implementations employing the multiread and window set functions.

Within the following discussion a series of terms associated with the field of data structures. References [42] provide an excellent discussion of the field.

The bintree is a well known image representation method [42] which expresses an image in terms of a hierarchy (or tree) of leafs. In terms of our notation (assuming the use of either the Morton or Peano order), each leaf consists of a continuous window set. If a leaf contains only light pixels it is called a white leaf. If a leaf contains only dark pixels it is called a black leaf. If a leaf contains both light and dark pixels it is called a grey leaf (or a node). Leafs are organized in a hierarchical structure known as a tree, each leaf may be split into two leafs (in precisely the same manner we split the

search space), the two resulting leafs are called children of the the initial leaf, and the initial leaf is called the parent of the the two resulting leafs. The leaf that contains the entire image is called the root and has no parent. Each black or white leaf terminates the descent of the tree, that is, such leafs do not have children. Each leaf may be given an address, one of the most common systems of addressing employs the same notation as the window set expression with the Ø digits deleted (for the sake of clarity we shall retain the Ø digits). Figure 2.10 illustrates the structure of a bintree.

It is possible to list the contents and structure of a bintree in two ways. The first method is to list the addresses of both the black and white leafs in order[42]. The second method is to list the color (black, white, or grey) of every leaf in prefix order[42]. The prefix order is a means of assigning a specific order to a hierarchical structure. In our notation, the prefix order may be defined as follows: leafs are visited in numerical order (replacing Ø with 0) , in the case of leafs of differing size the larger leaf proceeds the smaller leaf (i.e. 00ØØ < 000Ø < 001Ø < 1ØØØ). We may define the order of traversal recursively in the following manner, visit the root of the tree, traverse the tree beneath the first child in the prefix order and then traverse the tree beneath the second child in the prefix order.

There exists a broad range of image processing algorithms that operate upon the bintree[42]. The utility of algorithms that employ the bintree representation are twofold: The first is that the hierarchical structure of the bintree reveals the interconnection of the various regions of the image. The second is that expressing an image in terms of the leafs of a bintree results in a representation with far fewer elements than a pixel representation, reducing the volume of the data that must be manipulated. This second property also ensures that compact representations of the bintree, such as the two noted above if properly encoded, result in a compression of the image.

Given the definition of the bintree in terms of the window set notation we have employed, it is not surprising that the second algorithm may be modified to decompose images into bintrees. In fact, only two modifications of the second algorithm are required to obtain an algorithm which results in a bintree representation. The first modification is to limit the collapse of the l and d functions to window sets which are continuous. This may be achieved by examining the results of l and d and if the results are not continuous expanding them to a continuous window via the method outlined in section 2.2.1.5. This assures that all window sets employed within the algorithm are contiguous, since the two operations which modify the extent of window sets are the recursive split which already divides a window set into two continuous window sets, and collapses, which we have just limited to contiguous blocks. The second modification is to recognize that the difference of two continuous window sets results in two regions, a region which proceeds the window set which has been subtracted, and a region which follows it. When decomposing such a difference into individual windows as per section 2.2.1.5., it is necessary to place the components in the proper order. This requires that we retain the portion of the difference set which follows the window set which has been subtracted, so as to place it in proper order, that is, after the contents of the window set which has been subtracted. This may be achieved by the use of a stack.

The first of the above modifications, limits the preformance of the second algorithm by limiting the extent of the collapses that may occur. If the loss of performance is unacceptable, the original algorithm may be employed in conjunction with postprocessing to generate the associated bintree.

Although the bintree is a powerful means of image representation, it has one major failing, the leafs that compose it have three possible configurations: A square, a rectangle which is twice as broad as it is tall, and a rectangle which is twice as tall as it is broad. This considerably complicates certain image processing algorithms that employ the bintree. A close variant of the bintree, the quadtree, solves this problem by eliminating the rectangular leafs. For this reason, the quadtree is often more useful than the bintree.

We now define the quadtree in terms of window set notation. Like the bintree, the quadtree is composed of leafs, which are continuous window sets with a size which is a power of four, i.e. they have an even number of variable bits(Ø). Black, white and grey leafs are defined in the same manner as a bintree. Like the bintree the leafs of the quadtree are organized in a tree structure, with each leaf having a parent (except the root), and four children as opposed to two in the case of the bintree. Each leaf may be split into four children, this split may be achieved by applying our conventional split twice. Each leaf may be given an address identical to its window set expression. The prefix order is defined identically. The contents of the quadtree may be listed in the two fashions that a bintree may be. Figure 2.11 illustrates the structure of a quadtree.

An image expressed in terms of the black and white leafs of a quadtree has an interesting interpretation. Given that all leafs are squares with dimensions that are powers of two, it is possible

to interpret leafs as pixels of varying resolution. In this interpretation, the quadtree viewed as a representation of the image in which each feature is depicted with the coarsest resolution possible. This explains the utility of the representation, in that algorithms are not forced to deal with large numbers of pixels within large constant areas. The potential for compression is also readily apparent. An unexpected advantage of this method is that a tally of the number of leafs of each possible size results in a histogram known as the complexity spectrum which is useful in distinguishing various classes of images[45].

Another highly useful property arising from the variable resolution of the quadtree representation is that it is possible to store the pixels in order of coarseness. Such a so called progressive representation effectively first reconstructs a coarse resolution image and then progressively refines the resolution of the image. Such a representation has the utility of presenting an intelligible image of degraded resolution if the representation is truncated at some point. Such a truncation may be intentional, or as a result of errors in transmission or storage.

As with the bintree, the second algorithm may be modified to decompose images into quadtrees. Unlike the bintree case, the modifications required considerably more involved. Not only must the results of the l and d functions be constrained to results that correspond to quadtree leafs, but the recursive split operation and the decomposition of difference sets into window sets must be reformulated. We shall defer the discussion of the resulting algorithm to the following section.

As with the modifications adapting the second algorithm to bintree extraction, the modifications noted above reduce the performance of the second algorithm. Given the greater constraints imposed upon the collapse of the search space in this modification, the performance of this modification is also lower than that of the bintree variant. This loss of performance may be ameliorated in three possible ways: The results of the original second algorithm may be converted to a quadtree form, a potentially difficult task. The bintree variant of the second algorithm may be employed and the well known algorithm for the conversion of a bintree to a quadtree[42] may then be employed to convert the results. The bintree variant of the second algorithm may be modified to preform the conversion of the bintree to a quadtree as the bintree is extracted. This latter strategy is the most promising, but a practical formulation has not yet been achieved.

Although it is possible to employ the variants of the second algorithm to decompose an image into a quadtree representation and then process that result, it is also possible to perform image processing during or instead of decomposition. A large proportion of the algorithms which employ quadtrees operate by traversing the quadtree and gathering information or performing operations upon leafs in the process. The quadtree variant of the second algorithm essentially consists of an accelerated prefix traversal of the quadtree which is being constructed. It is possible to use the l and d functions and the method of recursive splitting to address the leafs of the quadtree of an image without explicitly extracting the quadtree. Since our methods address the image in a parallel associative manner there is no penalty in accessing the image directly. Such direct access algorithms may prove useful in tasks employ the quadtree representation to obtain a single specific result, such as an image outline, or the topological properties of the image.

2.4.3.2.4. Depth First Compression.

We now present an adaptation of the Depth First (DF) compression algorithm for quadtrees. The DF compression technique[45,46] simply encodes the results of a prefix traversal as a compact, variable length binary code. This method achieves lossless compressions on the order of 3:1 to 15:1 depending upon the content of image and may be extended to the lossy compression of grey scale images.

It is possible to describe a quadtree in terms of a combination of parenthesis and 0 and 1 values. The method is to enclose each progressive layer of the quadtree in a pair of parenthesis. For example we may consider the appropriate expression for the quadtree depicted within Figure 2.11. For clarity we compose the expression from the top down, denoting unfinished portions with a *:

(*111)
((0*0*)111)
((0(1101)0(0111))111)

Although such a description method is sufficient to fully describe the structure of an arbitrary quadtree, it is also redundant. Since the quadtree always branches into four elements, it is always the case that each expression enclosed by parenthesis contains four elements. Each element either consists of another expression within parenthesis or a simple value (0 or 1). Thus we may always predict the location of the closing parenthesis ")". This means that the closing parenthesis may be discarded without the loss of information. By doing so we obtain the DF expression for a quadtree. For example by deleting the closing parenthesis from the above example we obtain the DF expression for the quadtree within figure 2.11:

    ((0(11010(0111111

As noted above the DF expression may alternately be viewed as a prefix order traversal of the quadtree. In this case (denotes a grey leaf, 0 denotes a white leaf and 1 denotes a black leaf.

As noted in the previous section, the quadtree variant of the second algorithm may be viewed as an accelerated traversal of the quadtree associated with the image under consideration. The traversal varies from a the prefix traversal due to the presence of collapses. In the absence of collapses, the algorithm corresponds to a prefix traversal. If we record within a list a (for every split we employ, and a 0 or 1 for all black or white window sets (leafs) the list resulting from a decomposition is the DF expression. The presence of collapses complicates matters since each collapse may correspond to a rather complicated structure. We shall now discuss the reason for this.

The collapse function essentially skips layers which have only have a single grey leaf, and otherwise contain all white or black leafs. For example, figure 2.12 illustrates the quadtree associated with an image containing a single pixel. This image is decomposed immediately due to a collapse. In comparison, the associated quadtree is quite complicated. Thus a collapse may correspond to a descent of several layers along a branch of the tree. In order to enumerate the structure of the skipped layers, we employ the set difference operation. The difference operation generates the difference set in prefix order, thus corresponding to a traversal of the skipped layers. Thus the set difference operation may be used to generate the DF expression associated with a collapse.

In order to generate the DF expression associated with a collapse, we must understand how the difference set corresponds to a prefix traversal. We may view the difference set operation, as a series of recursive splits of the inital window set, where at each split, each of the three resulting window sets which do not contain the window set which is being subtracted are added to the difference set. Following the example of figure 2.12, we first split the initial window (the entire image) into four parts (quadrants of the image) and add the first three windows into the difference set. After the second split we add the last three windows into the window set. With the final split, we add the first two windows, and the last window to the difference set.

Following our example, we compose a parenthesis list of the quadtree from figure 2.12.

    (111*)
    (111(*111))
    (111((1101)111))

by deleting) characters we obtain the equivelent DF expression

    (111*
    (111(*111
    (111((1101111

As we may see, at each layer of descent, we add one (followed by three 0 or 1 values (whichever value forms the background). Up to three of the latter values may occur after the point at which the list is updated (this point is designated by a *). This complicates the formation of the DF expression, since all of the previous operations have sequentially added to the DF expression. Handling collapses, on the other hand, requires that information be written out of sequence. This problem may be corrected for by storing the values in question on a stack, allowing us to defer the adding the values until the point at which the values would be written in sequential order.

We now present the algorithm for DF compression employing the multiread and window set operations. As with the second algorithm, the variables r and v denote window sets, while the l and d functions are the same as in the second algorithm. We employ a simple FIFO (First In First Out) stack, which we call a write ahead stack, to handle non-sequential writes to the DF expression as a result of collapses.

1. Initialize both r and v to include the entire search space. The DF expression is set to blank.

2. Determine the values of l and d for the current value of r. For both l and d preform the following operation. Locate the most significant occurrence of a Ø value, set all less significant positions to a value of Ø (this limits results to continuous window sets). If there are an odd number of Ø values afterwards, set the least significant constant bit (0,1) to a value of Ø. (this limits results to square continuous window sets, e.g. quadtree leafs).

3. If l is a null set then proceed to step 8. Append a value of 0 to the DF expression.

4. If d is a null set then proceed to step 8. Append a value of 1 to the DF expression.

5. If d is a subset of r then set the value of r equal to the value of d. Set the background value to 1. Call the set difference subroutine. Proceed to step 2 unless d contains no Øs. If d contains no Øs, append a 0 to the DF expression, proceed to step 8.

6. If l is a subset of r then set the value of r equal to the value of l. Set the background value to 0. Call the set difference subroutine. Proceed to step 2 unless l contains no Øs. If l contains no Øs, append a 1 to the DF expression and proceed to step 8.

7. Find the two most significant positions within r with a value of Ø and replace both with 0 values. Replace the same positions within v with 0 values. Proceed to step 2. Append a (to the DF expression. Push four blank values "" onto the write ahead stack. (This amounts to a recursive split, which splits values into quadtree leafs)

8. Pop a value off of the write ahead stack and append it to the DF expression. Split the expression for v into pairs of values, find the least significant pair of constant values which is not equal to 11. Increment this pair by one and replace (i.e. 00 -> 01, 01 -> 10, 10 -> 11). Proceed to step 2. If all constant values are set to 1 then Halt. (This is a simple termination of recursion which takes into account quadtree leafs.)

Set Difference Subroutine.

1. Set a=r. Set c=0

2. If the background value is 0 then b=l otherwise b=d.

3. Compare a and b. Form a list of the positions within a have constant bits and the positions of b have variable bits. Split this list into pairs. For example:

    a= 1010110Ø
    b= 10ØØØØØØ
      1011
      10,11

4. Examine each pair in order of significance. For each pair append the appropriate table entry to the DF expression and add the appropriate increment to c.

Pair value	Append to DF exp.	Increment c by
00	(	3
01	(x	2
10	(xx	1
11	(xxx	0

x denotes the background value (0 or 1)

5. Push a string of c background values into the write ahead stack. Return to calling procedure.

This algorithm is essentially a straightforward variant of the second algorithm, modified to employ only quadtree leafs. As noted in the previous section, quadtree leafs must be split into four leafs rather than halves, thus our methods for recursive splitting and recovery must operate with fourfold splits, rather than the twofold splits employed within the second algorithm. Steps 8 and 9 achieve this by operating upon bits in pairs. In the same fashion, the additions to Step 2 ensure that l and d may

only collapse to quadtree leafs. This eliminates the possibility of l and d both collapsing, since both would collapse to quadtree leafs, each of which only fill only one quarter of the search space, leaving one half of the search space neither light nor dark. Thus we have eliminated steps to handle the joint collapse of both l and d.

The set difference subroutine operates in the same fashion as the original algorithm in section 2.2.1.5. but employs bit pairs to ensure that the results consist of quadtree leafs. As we have noted a write ahead stack is employed to maintain sequential writing of the DF expression. This stack is pushed once during the set difference operation since the section written ahead merely consists of a series of background bits of the same value. These values are popped and written each time recursion terminates. Recursive splits push dummy values to the same stack to prevent the premature writing of the data. As with the second algorithm the actual recursion is achieved implicitly and without the use of stacks.

This algorithm has been successfully implemented in software, operating with serial implementations of the multiread and window set function. The output and compression rates achieved are identical with the original serial algorithms employed. Apendix A indicates the performance in cycles and compression rates for the several test figures illustrated.

In successfully implementing the DF compression method with a augmented version of our second algorithm we have illustrated both an algorithm which is useful in its own right, and a strategy for the implementation of a broad range of potentially useful image processing algorithms.

2.4.3.2.5. Other Means of Image Compression

The next method of image compression that we shall consider is the direct use of the second algorithm for image compression. As noted in previous sections, the bintree and quadtree variants of the second algorithm suffer from reduced performance in comparison with the original algorithm due to constraints upon the possible collapses that may occur. Thus if a sufficiently compact coding of the list of window sets and difference sets is possible, the use of the original second algorithm would be preferable.

The second algorithm, as previously described generates a list of light pixels in the form of window sets and window set differences. Although such a list is less redundant than a serial list of address, it still contains considerable redundancy. This redundancy arises from the fact that two subsequent entries within the output will often only differ by several bit positions. This is to be expected, given the physically adjacent nature of the two entries. Therefore it should be possible to further compact the output of the second algorithm.

The most promising method for obtaining a more compact output is to generate a compact history of the program flow during an image decomposition. Since such a history would allow us to reconstruct the operation of the program, we would be capable of regenerating the output list generated by the program, or directly regenerating the image itself.

In order to ensure maximal compression it is necessary to record the minimal information for each cycle of the program. Thus we must examine each possible outcome and determine the most compact means of recording the results.

In the case of cycles that result in recursive splits, it is only necessary to note that a split has occurred, since the order of splitting is known and no other change to the program occurs.

In the case of cycles in which the search space is either all dark or all light (either l or d return the null set) it is only necessary to note whether the window is light or dark and to note the termination of recursion. It is not necessary to specify the window set, since the extent of the search space r may be inferred from the previous sections of the program history.

In the case of a collapse of either the l or d function it is necessary to note which function collapsed and the resulting window set. By necessity, the constant bits within the search space r have the same values as the same positions within the collapsed set. Since the state of the search space is already known during reconstruction, the common constant bits between r and the collapsing function are redundant and may be discarded. In addition, if several collapses occur sequentially, it is not necessary to note the function which collapses for those collapses which occur after the first. This is because, as we have noted in section 2.3.3, sequential collapses must alternate between the l and d functions.

Finally in the case of a collapse of both the l and d function, it is sufficient to treat this circumstance as a combination of the collapse of one of the functions, followed by a filled search space.

We now present a modification of the second algorithm which implements this strategy. As with the second algorithm, we shall use the two window sets r and v the former describing the search space, while the latter is used to track the progress of recursion. It is important to note that r is always a subset of v. We also introduce a boolean flag COLL which records whether a collapse had occurred in the previous cycle.

1. Initialize both r and v to include the entire search space. Set the value of COLL to false.

2. Determine the values of l and d for the current value of r.
3. If l is a null set then proceed to step 9. Record a termination of recursion. If COLL is false record a background value of white (1). Set the value of COLL to false.

4. If d is a null set then proceed to step 9. Record a termination of recursion. If COLL is false record a background value of black (0). Set the value of COLL to false.

5. If l and d are disjoint sets then proceed to step 9. Record that a collapse has occurred. Record a backround value of black (0). For each variable bit (Ø) position within w, record the contents of l at that position. Record a termination of recursion. Set the value of COLL to false.

6. If d is a subset of r then set the value of r equal to the value of d. Record that a collapse has occurred. If COLL is false then record a backround value of white (1). For each variable bit (Ø) position within w, record the contents of d at that position. Set the value of COLL to true. Proceed to step 2 unless d contains no Øs. If d contains no Øs proceed to step 9 and record a termination of recursion.

7. If l is a subset of r then set the value of r equal to the value of l and proceed to step 2 unless l contains no Øs. Record that a collapse has occurred. If COLL is false then record a backround value of black (0). For each variable bit (Ø) position within w, record the contents of l at that position. Set the value of COLL to true. If l contains no Øs, append the value of l to the list of true values and proceed to step 9 and record a termination of recursion.

8. Find the most significant position within r with a value of Ø and replace it with a 1. Replace the same position within v with a 1 Proceed to step 2. Record that a split has occured. Set the value of COLL to false.

9. Find the least significant position within v containing a value of 1 and replace it with a 0. Set the value of r equal to that of v. proceed to step 2. Set the value of COLL to false. If there are no remaining values of 1 within v Halt.

The contents of the compressed signal consist of an alphabet of six characters, three data characters (0,1,Ø) and three characters which denote, respectively, splitting, collapse and termination of recursion (s,c,t). Such a large number of characters (as opposed to the three characters required for DF encoding (0,1,()) requires a more complex encoding scheme, which may limit the compression rate of the algorithm. Employing fixed length codes leads to results that display negligible compression. Ad hoc variable length codes have resulted in modest rates of compression (3:1-5:1). Given that each of the characters of this alphabet have radically different probabilites of occurrence within the output, Huffman coding is an attractive option. A fixed Huffman code could be simply implemented and could promise considerable compression rates. Another promising option is to combine variable length coding with run length encoding, since several characters are often repeated within the output, specifically the s and Ø characters. Although the algorithm described above would appear to be a more elegant application of the second algorithm to the task of image coding than DF encoding, the complexity of coding the output may limit the possible compression rate possible. Clearly, further research is required.

2.5. Implementation of Our Algorihms

2.5.1. Introduction

In the previous sections we have defined the multiread function, the window set function, and the Lewin algorithm and its varian s. In this section we describe the methods by which we may

implement these algorithms upon a special purpose optical computer architecture based upon the HOPLA. We shall also discuss how various physical sources of error may be addressed by the modification of our designs. We shall also consider how error control and graceful degradation may be integrated into our systems.

2.5.2. Implementation of the Multiread Function

Previously, we have formed definitions of the multiread function and the window set function in terms of boolean matrices. As we have seen in chapter 1, such boolean matrix formulations may be directly translated into control state operator implementations. Given that the HOPLA is a control state operator system, it is quite simple to translate our formulations into interconnect patterns and control masks.

Our strategy is as follows. We shall use the two dimensional mapping of the multiread function formulated for image compression, to form a mapping from the input plane to the output plane of a single stage implementation of an HOPLA. Such a single stage implementation is sufficient to generate the multiread function. Once the interconnect pattern has been devised, it is then possible to transform them into the appropriate control masks necessary to construct the desired HOPLA.

We have previously defined the multiread function in terms of boolean vectors as follows:

$$\underline{w}^0 = [S \, \underline{p}]'$$

$$\underline{w}^1 = [S' \, \underline{p}]'$$

Where the matrix S is defined as follows:

$$k = \sum_{i=0}^{m-1} 2^i S_{i,k}$$

The output $\underline{w}^1_{i,k}$ is connected to every input p_k for which the ith bit of the index k has a value of 1. In the same fashion, the output $\underline{w}^0_i$ is connected to every input p_k for which the ith bit of the index k has a value of 0. Thus the interconnect pattern for a specific input pattern is purely dependent upon its index. The S matrix simply provides a convenient mathematical expression of this statement.

From this formulation, we may make several important observations: There are 2^m inputs and 2m outputs. Each of the inputs is connected to half of the outputs, half belonging to the $\underline{w}^1$ vector and half to the $\underline{w}^0$ vector. Each connection pattern is unique, and all possible permutations are covered. Conversely, each of the outputs are connected to half of the inputs, and any two inputs (except those which are complementary, i.e. $\underline{w}^0_i$ and $\underline{w}^1_i$) have a quarter of the inputs in common.

This inteconnect pattern must now be mapped into two dimensions. We exchange a 2^m long input vector p for a $2^{m/2} \times 2^{m/2}$ input matrix I. The matrix I corresponds to the contents of the input matrix of the HOLPLA. Given that in general the physical positioning of the outputs has no geometrical significance, we shall not attempt to map the output in the same manner but instead continue to treat the outputs by their name and index. Given a specific mapping k(x,y) of the indices x,y to k we may then directly obtain the two dimensional interconnect pattern:

$$w^1_i = [\sum_i \sum_j S_{i,k(x,y)} I_{x,y}]'$$

$$w^0_i = [\sum_i \sum_j S'_{i,k(x,y)} I_{x,y}]'$$

Note that the S matrix behaves as a three tensor in this case since its second index has been mapped into two indices. The corresponding control masks are simply the matrix formed by the fixing the value of i. Thus the control mask for the output w^1_1 is simply the matrix formed by $S_{1,k(x,y)}$, while the complement of this matrix forms the control mask for the output w^0_1.

Previously we have discussed three different mappings into two dimensions, the raster scan order, the Morton order and the Peano order. As we have noted previously, the raster scan order differs from the Morton order only in the designation of the outputs, while the pattern of the interconnects and the control masks employed are identical. We now consider the interconnect patterns resulting from these orders.

Since it is nearly impossible to illustrate the three dimensional nature of the full interconnection pattern it is best to illustrate the interconnection pattern in terms of the resulting control masks. The control masks indicate the set of inputs which connect to the given output associated to the mask.

2.5.2.1. The Raster Scan and Morton Orders.

The raster scan order is the simplest order to describe, the mapping is quite simple:

$$k(x,y)=2^{m/2}y+x$$

The control masks associated with this mapping are quite simple. Figure 2.13. illustrates the control masks associated with $\underline{w}^1$ for the case m=16. As the illustrations indicate, each the control masks consist of equally spaced bands of black (0) and white (1) bars with widths which are a power of two, oriented in either the x or y direction. The bar widths range from 1 to $2^{m/2-1}$ pixels wide. The Morton order has identical control mask patterns, except that the order of the control masks is interspersed in the manner described in section 2.4.3.1.2.

There are two major merits to this control mask pattern. The masks are exceedingly simple to generate and model, allowing for simple verification and analysis. Secondly each of the control masks are separable functions, only displaying modulation along one axis. This gives us the option to generate the control masks using apertures that sacrifice resolution along the axis which is not modulated.

Unfortunately these patterns also have a major disadvantage as well. They have sharply peaked autocorrelation functions as a result of their periodicity. Such patterns are poorly suited for recording by Fourier transform holography, since the broad dynamic range of the object beam leads to a broad variance of the object beam reference beam ratio[47]. Although this problem may be addressed by various strategies that we shall consider in later chapters, it still complicates implementation.

Another disadvantage of the Morton order is that for a given input pixel, the number of control masks at which that pixel is on the boundary of a region, varies considerably depending upon the pixel position. For example the four central pixels are on the boundary of a region for all of the control masks, while there exist pixels which are on the boundary of a region for only four of the control masks. Since the boundaries of regions within control masks are the areas most prone to crosstalk, some input pixels are potentially far more prone to crosstalk than other pixels. It would be preferable to have a system of control masks over which the number of boundaries encountered by each pixel is the same. A connection scheme which addresses this problem is made possible by the use of Gray code.

A Gray code is a binary numbering scheme which has the property that the hamming distance between any two values is always one. There exist several different codes of this form. It is always possible to convert a conventional binary sequence into a "canonical" grey code by simply preforming a bitwise exclusive or (XOR) between the conventional value and the conventional value shifted right by one. We may perform this transformation upon the S matrix in order to obtain a grey coded version we shall denote S^*

$$S^*_{mj}=S_{mj}$$
$$S^*_{ij}=S_{i+1,j} \text{ XOR } S_{ij} \quad i=1...m-1$$

If we employ the S^* matrix instead of the S matrix within our interconnection scheme, we obtain a different set of control masks. These control masks are quite similar to the control masks used in the conventional morton order except that they are symmetrical about the center. These control masks have two useful properties: Each pixel is on the boundary of a region in only two control masks. This minimizes the effect of crosstalk, and distributes evenly the vulnerability of each pixel to crosstalk.

The second feature is that the highest resolution of the control masks employed is half that of the non-Gray coded control masks.

The Gray coded mask system essentially corresponds to a different mapping of the p vector and the I matrix. Being such, it functions equally well as an implementation of the multiread function. In simulations, this order functions equally well as the Morton or Peano order for the decomposition of images, and for DF compression.

2.5.2.2. The Peano Order.

The implementation of the multiread function employing the Peano order mapping is somewhat more complicated. The mapping function $k(x,y)$ is defined by the use of a well known algorithm devised by Butz [37]. Given this algorithm we may generate the control masks for any given value of m. Figure 2.14 illustrates the control masks for the case m=16. As we may see the control masks alternate between checkerboard patterns and irregular (although highly symmetric) patterns. The square size for the checkerboard masks are powers of two.

Unlike the Morton order, the control masks associated with the Peano order are not separable, nor easily subject to analysis. In fact, the patterns resulting from the non-checkerboard masks are fractal. This follows from the fact that the Hilbert curve is fractal in nature [48], and the control masks consist of a mapping of a periodic function along this curve.

Although fractal structures display a high degree of self similarity[48], their autocorrelation functions are considerably smoother than simply periodic functions. For this reason, control masks displaying a fractal structure would be well suited for Fourier holography, due to their limited dynamic range. Unfortunately, the checkerboard pattern control masks are highly periodic and thus ill suited for fourier holography.

A Gray coded version of the Peano order is also possible, and its control masks are displayed in figure 2.15. The most interesting property of the resulting masks is that they are all irregular fractal patterns, and therefore well suited to Fourier transform holography. Simulations indicate that this order is well suited for both the implementation of the second algorithm and the DF compression algorithm.

2.5.3. Implementation of the Window Set Function

In section 2.2.1.4 we presented a formulation of the window set function in terms of boolean matrices:

$$\underline{q}(w)=[(S^T \underline{w}^0) \text{ OR } (S^T \underline{w}^1)]'$$

The interconnection pattern associated with this equation is nearly identical with the interconnection pattern required for the multiread function. The first difference that the inputs and outputs have been exchanged. The second difference is that the control mask associated with the output $\underline{w}^1_i$ in the multiread function is now associated with the input $\underline{w}^0_i$, just as the control mask associated with the output $\underline{w}^0_i$ is now associated with the input $\underline{w}^1_i$.

It is possible to implement the window set function by employing a variant of the single stage HOPLA architecture that implements the multiread function. Within this variant, illustrated in figure 2.16, an SLM containing the input is placed in contact with the POHM which normally mediates the interconnection, and the input of an inverting Optical SLM (OSLM) is located at the position normally occupied by the input SLM in the conventional N^4 interconnect. Each pixel of the input SLM within this system covers a facet of the POHM. Each of these pixels is associated with the appropriate input value ($\underline{w}^0$ or $\underline{w}^1$) and is set to a clear state when that input is true, and an opaque state otherwise, projecting the appropriate control mask upon the OSLM. The OSLM takes the superposition of the control masks projected upon it and inverts the values, generating the appropriate window set at the output. One of the advantages of this system is that the interconnection POHM is identical to that used to implement the multiread function.

2.5.4. Implementation of Lewin's Algorithm and its Variants.

We may divide the implementation of Lewin's Algorithm and its variants into two parts; the implementation of the l and d functions, and the implementation of a logical control unit. The latter

task merely consists of of the construction of a relatively simple finite automaton which serves to store values and control program flow. In comparison, the practical implementation of the l and d functions requires large scale global parallelism of the sort devised in this text. For this reason we shall concentrate on the implementation of the l and d functions.

It is only necessary to implement both the l and d functions for our second variant of Lewin's algorithm. If we only wish to implement Lewin's original algorithm only the l function is required. In the same sense, it is only necessary to implement the l and f^0 functions in order to implement our first algorithm and Lewin's original algorithm. Thus partial implementations of the l and d may still retain some utility.

2.5.4.1. Implementation of the l and d functions.

In section 2.3.3 we defined the l and d functions in terms of their component values l^0, l^1, d^0 and d^1:

$l^0(w,p) = [S\ q(w)^T p]'$
$l^1(w,p) = [S'\ q(w)^T p]'$
$d^0(w,p) = [S\ q(w)^T p']'$
$d^1(w,p) = [S'\ q(w)^T p']'$

Where is $q(w)$ the worst case window set function:
$q(w) = [(S^T w^0)\ OR\ (S'^T w^1)]'$

Given that we have expressed the four required functions in terms of boolean matrix functions it would be possible to implement the functions by the use of four independent two stage HOPLAs. Such an implementation would be quite wasteful, since a considerable degree of redundancy and symetry exists between the four functions we wish to implement. By exploiting such features we may construct an implementation which is much more efficient both physically and in terms of reduced interconnection complexity.

The first step in implementing the desired functions is the generation of the worst case window set function $q(w)$, which is common to all four functions. As we have shown in section 2.5.3. the $q(w)$ function may be implemented by the use of an appropriately illuminated POHM which projects selected control masks upon an OSLM which thresholds and inverts the projected maps, performing the NOR function. We shall refer to this OSLM as the dissection SLM in that it its function is to block the portions of the of the p vector that have been eliminated through splitting or collapse.

The second step in implementing the functions is to AND the q function with the p and p' vector. Such a function may be performed by optically superimposing the dissection SLM and the input SLM which contains the p vector. We may evaluate both the $q^T p$ and $q^T p'$ products simultaneously by superimposing an amplitude modulated dissection SLM upon a polarization encoded input SLM and dividing the resultant signal into the resulting orthogonal states of polarization. Such a scheme has the advantage of simultaneously generating both results while employing the same hardware and energy that would be used to generate either of the single products if a pure amplitude scheme were used.

The final step is to perform the multiread function upon the $q^T p$ and $q^T p'$ products in order to obtain the l and d functions respectively. Within section 2.5.2. we have described how the multiread function may be implemented by the use of the multiread function. Such a method may be used to generate the l and d functions simultaneously by employing the polarization encoding scheme described within the previous paragraph. Within such a scheme the conventional N^4 interconnect would be modified so as to pass through both the dissection and input SLMs, and then pass through a polarization beamsplitter. The two resulting outputs would correspond to the l and d functions respectively. Ideally such a scheme would best be implemented by mounting the dissection and input SLMs in close contact. Unfortunately, it is unlikely that such a strategy would prove practical due to the structure of most SLMs. Thus the the more practical method would be to optically superimpose the two SLMs via the use of a relcy lens system. Such a scheme is illustrated in figure 2.17. In this scheme, a reflection type N^4 interconnect evaluates the dissection matrix which contains the q vector, the resulting output rather than being detected is then relayed through a second N^4 interconnect configuration containing the input SLM which contains the p vector which is polarization encoded.

The structure of the relay lens system is similar to the lens system used within POHM systems that do not employ conjugate reconstruction[49]. The resulting signal is then split by a polarization beamsplitter and the two resulting wavefronts focus upon separate detector arrays. The two resulting array outputs correspond to the l and d functions respectively.

The proposed implementation described above has been devised in such a manner as to minimize the number of active components while maximizing the degree of interconnection. Within both stages, each input is connected to half of the possible outputs. Within the second stage all of the light which strikes the input SLM contributes to either the l or d function (excepting the portion lost to attenuation, scatter, diffraction, etc...). Unfortunately, although elegant by the above considerations, the proposed implementation may prove to be too intricate to be practically viable. Such a scheme would of course be highly sensitive to misalignment of components as well as the culmative distortions involved in passing through a double interconnect system. For this reason we shall now discuss how we may simplify the configuration of the system at the cost of increasing the number of active elements.

The simplest alternative implementation would be to explicitly compute the two products $q^T p$ and $q^T p'$ and then evaluate the multiread function for each product independently in order to obtain the l and d functions respectively. Such a strategy is somewhat related to the "dual rail" scheme described in chapter 1, but differs in two respects; the two products are not true complements, and the two products never appear together in the same logical expression. The advantage of this strategy is that the evaluation of the multiread function proceeds in the same fashion as described in section 2.5.2. employing a conventional HOPLA architecture. The disadvantages of this strategy are that a greater number of active elements will be required, since it will prove necessary to latch both the values of q and p separately and then form the two desired products upon a third OSLM. If the two products are generated simultaneously third OSLM will have to contain twice the number of active elements as the size of p. If, on the other hand, the two terms are generated and evaluated sequentially, then the number of cycles required for evaluation doubles. Of course if the system described is being employed as an adjunct to an associative memory or an image processing system the hardware to explicitly evaluate the desired products may already exist.

The other alternative implementation of the l and d is to employ a folded version of the the interconnection architecture which shall be discussed in section 2.6. This strategy abandons the use of the HOPLA entirely while employing a two stage architecture with a greatly reduced level of fan-in. Although this strategy may prove more practical for implementation by current active devices it is much less flexible and global in the structure of its interconnections.

2.5.4.2. Implementation of the Logical Control Unit (LCU)

Within this section we shall loosely sketch the implementation of the Logical Control Unit (LCU). Our description shall not be as comprehensive as that of the l and d function, since the LCU may be easily implemented employing conventional logical components. In fact it is assumed that the output, consisting of the values w^0 and w^1 and the detected input values l^0, l^1, d^0 and d^1 take the form of electronic signals and are processed by an electronic logical control unit. This is due to the fact that the great majority of the logical operations involved within the execution of Lewin's algorithm are involved in the evaluation of the l and d functions which demand highly parallel architectures to implement. In comparison, the operations performed by the logical control unit consist of a relatively small number of simple logical operations which are easily and quickly performed by electronics. Unless a practical advantage in speed were possible, no advantage would come of implementing the LCU optically.

We may divide the functions of the LCU into four major categories, storage, processing, program control, and housekeeping. The LCU must store the values of the window set variables r and v as well as latching the input and output values l,d and w. In terms of processing, the LCU must be able to perform bitwise logical and comparison operations between its various registers. The program control consists of a finite state automation which directs the operation of the processing section and employs the results and its previous state to determine the output generated by the LCU. Finally the LCU must generate the appropriate hardware control signals necessary to sequence the operation of the l and d function evaluation hardware, a set of tasks that we shall refer to as housekeeping. Figure 2.18 sketches the proposed structure of the LCU.

The task of storage is the most straightforward. As we have noted above, it is necessary to implement registers to store the variables employed within the algorithms (r and v), as well as latch

the input and the output of the LCU (l,d and w). We may implement these window set registers as pairs of binary vectors (or words), employing the "x^0,x^1" encoding we have previously formulated. It is also necessary to add several temporary window set registers to hold the result of operations and comparisons between registers. Finally, a pair of "constant" registers containing all true and all false values should be implemented for convenience.

The Logical Processing Unit (LPU) of the LCU corresponds to the Arithmetic Processing Unit (APU) of a conventional CPU, in that it preforms operations and comparisons between registers. The LPU differs from a full blown APU in that it is considerably simpler since it only performs bitwise logical operations and several special purpose functions.

The simplest functions that the LPU preforms are the bitwise logical operations. The LPU must be able to perform the AND, OR, XOR and NOR functions between any two window set registers as well as between the binary vector pairs that compose each register. As with a conventional APU after every operation the LPU should set a flag indicating whether the result register is zero (all false) allowing for the LPU to be used in comparison operations.

The other function that the LPU must preform is to strip the all but the most significant true bit of a binary vector (which we shall call the STRIP operation). This function is employed whenever it is necessary to select the most significant bit in which a given value is located. The result of this function is a mask which may be used to "toggle" the most significant bit to a new value by the use of the bitwise XOR function. This function may be implemented by the use of a cascade of logic gates that propagate downward a supression signal activated by the first true bit. Such a logic circuit is often known as priority logic, and is employed in associative memories (that do not employ Lewin's algorithm) to suppress all but the first (with regards to address) responder to a search. Such circuits may be implemented quite simply, but operate more slowly than bitwise operations.

Finally, within some variants of our second proposed algorithm, it is necessary to limit the range of possible window sets to those which generate continuous addresses. Within section 2.4.3.2.4. we have described an operation (which we shall call the LIMIT operation) that maps any given window set into the smallest continuous window set which is a superset of the original. Such an operation may be implemented within the LPU as a hardwired logic circiut similar in form to the STRIP operation.

The LPU described above is should be sufficiently general to implement both Lewin's algorithm as well as all of the extensions and variants proposed within this chapter. If only one algorithm is to be implemented the design of the LPU may be considerably simplified by eliminating whatever operations, data paths and registers are not needed. The speed of such a specialized LPU may be enhanced by hardwiring the bulk of the required operations. The cost of such a design is increased gate redundancy.

The program control unit most closely resembles the control logic of a conventional CPU. The program control unit directs the flow of data to and from registers and controls the operation of the LPU in response to inputs generated by the comparison operations of the LPU. In essence, the program control unit contains the "program" that implements the desired algorithm. Such a "program" more closely resembles a single complex machine code instruction than a machine language program in memory. The closest analogy is to microcode, the program hardwired within a CPU which implements its machine language.

The actual form of the program control unit is dependent upon the degree of flexibility required. If the implementation of only a single algorithm is desired, the program control unit may simply consist of a hardwired finite state machine. If, on the other hand, several different algorithms must be implemented, the use of a simple program based system may be employed to achieve greater flexibility.

Of course, the explicit design of the program control unit is dependent upon the structure of the LPU implemented. If a single algorithm is to be implemented, the simplification of both the LPU and the program control unit will be considerable. In fact, the two units will be so closely linked as to be a single circuit. On the other hand, a flexible control unit would require a flexible LPU as well.

The task of generating the output for the system is also handled by the LCU. The output generated by Lewin's algorithm and our proposed variations consists of a stream of addresses (Lewin's algorithm), window sets (our first algorithm, and differences of window sets (our second algorithm). Additionally, several status values must be generated to communicate program flow and completion. The former task may be achieved by directing the contents of the l and/or d latches to a pair of output

latches. In the case of an output consisting of a window or an address (which may be seen as a special case window) a single latch is employed and the appropriate value is routed to it. In the case of a window set difference, a pair of latches are employed, denoting the two window sets involved. In this case both the values of l and d are employed, the order being dictated by the order of the difference. The latter task, generating status values, is handled by the program control unit directly. At each change of state, the control unit generates the appropriate set of status signals and activates the appropriate single bit output latches.

The task of housekeeping is quite similar to that of output and basically consists of generating the appropriate signals to sequence the operation of the system. The task is best performed by the LCU, since it also controls the program flow. As with output, the program control unit should perform the task of generating the housekeeping signals.

Having described the functions of the LCU we now consider several practical methods of implementing the functions of the LCU. For reasons previously noted, we assume that the LCU shall be implemented electronically. Ideally, the LCU should consist of a simple custom, special purpose processor. Obviously, the construction of such a unit is (at least initally) highly unlikely. Thus, it is necessary to construct the LCU from off the shelf components. The basic issue in such implementations is how much of the implementation takes place in hardware and how much takes place in software.

One practical design strategy would be to employ a programmable logic device (such as a PLA or a Field Programmable Gate Array (FPGA)) to construct a LCU. Such a solution although slower and less elegant than a fully custom design, is eminently practical and inherently flexible. Such an implementation would be optimized for a single algorithm since it would be possible to design LCUs for each algorithm.

Another design strategy would be to program a conventional microprocessor or microcontroller to emulate the desired LCU. Such an implementation would be slower than a hardware oriented implementation, but would be considerably more flexible. Since the primary loss in speed would arise from non-standard operations (such as the STRIP and LIMIT operations), adjunct hardware which preforms these operations may be used to accelerate operation. Given that the program to emulate the LPU is relatively small, and only employs a small number of registers, very little memory is employed for either the program or the operations. At worst, a small stack must be employed. Thus, it would be quite feasible to implement the required program employing the ROM and scratchpad RAM resident within most single chip microcontrollers.

The final option would be to eliminate the LCU entirely, and to directly access the l and d evaluation hardware. Such a scheme would be similar to architectures that employ associative memories as an adjunct to conventional Von Neumann designs. Such a design would maximize flexibility at the cost of speed. The primary advantage of such a design is that it could be used for applications of the multiread function other than ordered retrieval, such as those described within sections 2.4.1. and 2.4.2.

2.6. Folded Implementation of Lewin's Algorithm

2.6.1. Introduction.

Previously, we have shown the utility of our extensions upon Lewin's Algorithm for flag algebra and image decomposition and compression. We have also shown how the two critical functions required to implement these algorithms, the window set (or generalized addressing) function and the multiread function, may be implemented using single stage Holographic Optical Programmable Logic Arrays (HOPLA). The advantage of such implementations is that they fully exploit the potential of the HOPLA for global parallelism, result ... tentially immense throughput. Unfortunately, such highly interconnected systems require ... ns which are beyond the capacity of current active optical components to deliver. Thus it ... desirable to devise implementations which result in reduced fan-ins while retaining hig... vea ... th... ghput.

We now present a strategy for folding the multread function into a two stage OPLA which reduces the fan-in to the square root of the fan-in required for our single stage systems. Additional advantages of this strategy includ · the use of a simple architecture for the first stage and a scheme for enhancing the parallelism of our algorithm.

2.6.2. Two Dimensional Folding

The basis of our folded scheme arises from the fact that when employing the Morton or raster scan order with the two dimensional multiread function, the results are separable. That is, a given bit within the result is either entirely dependent upon the distribution of the x or y coordinates of the flags involved, but not both. Thus if we were to OR the values of each row and column of the image matrix together and perform the one dimensional multiread function upon the the two resulting "projection" vectors and then concatenated the two results together, we would obtain the multiread function for the the entire image matrix. Such a strategy clearly requires two stages, the generation of the projection vectors and their storage within a thresholding optical latch, and the extraction of the multiread function from the resulting vectors.

One way to interpret the use of row and column projections within the folded interconnection scheme is to view them as primitives that may be used to compose the required interconnection masks. Figure 2.19 illustrates the Morton order control masks for a 8x8 input matrix and a pair of arbitrary row and column projections. It is obvious that it is possible to compose all of the masks by combinations of either row or column projections. Thus the first stage extracts the required primitives and the second stage composes them into the desired control masks.

The advantages of this strategy are obvious, the fan-in required for each of the first stage elements is simply the size of the row or column, and for the second stage, half that number. In comparison our single stage scheme requires a fan-in corresponding to half the size of the input matrix. Thus, the fan-in in the original scheme is directly proportional to the number of elements within the image matrix, whereas within the folded scheme the fan in is proportional to the square root of the number of elements.

The price paid for the reduction in fan-in is threefold, two stages of interconnection are required, many more active devices are required and a reduction in the level of interconnection results in a proportional reduction of the throughput of the system. The increase in the number of active devices is the most easily quantified of these drawbacks. Since the outputs of the original and folded schemes are identical, both the single stage and the second stage of the folded system employ the same number of active devices. Thus the increase in the number of active devices within the folded scheme is due to the units within the first stage. The number of active units within the first stage is identical to the number of rows and columns within the image matrix which is proportional to the square root of the number of elements within the matrix.

2.6.3 Implementation of the Folded Scheme

The implementation of the folded scheme is of a considerably different form than that of the original scheme. The interconnections required for the first stage may be implemented with a pair of perpendicular cylindrical lenses as per figure 2.20. The resulting pair of projections are then thresholded by a pair of one dimensional OSLMs, which serve as the input to the second stage. The second stage unit simply preforms the multiread function upon the one dimensional input by means of an optical matrix-vector multiplication architecture followed by a threshold and invert operation. The fan-in of the second stage is equal to half of the length of the vector evaluated. Assuming a square input matrix, this is equivalent to half the fan-in of the first stage.

It is possible to employ such a scheme to evaluate the l and d functions by taking the projection of the input matrix and its complement and performing the multiread operation upon the respective results. Such a scheme may be implemented by the use of polarization optics in the same manner as has been suggested for our original design. Implementation is simplified by the fact that a single collimated beam propagates through the input and dissection SLMs. Although it would be possible to use a folded scheme to generate the window set function, the simplicity, and relatively low fan-in of of the original design may well prove more practical.

2.6.4. Parallel Evaluation Employing the Folded Architecture.

An unexpected advantage of the folding scheme, is that it may be used to increase the parallelism of Lewin's algorithm and its variants. This arises due to the structure of the projection functions involved. Figure 2.21 illustrates several window functions which have projections which do not overlap. Given the lack of overlap, it is possible to to employ a folded architecture to simultaneously evaluate the first stage values of all of the illustrated windows simultaneously. The multiread function may then be evaluated for each pair of projections associated with an individual window set. Although the multiread function must be applied individually to each individual projection, all of the individual functions may be evaluated in parallel by an appropriately configured second stage. Since

the the second stage of logio consists of a vector-matrix multiplier architecture, it is possible to reconfigure the interconnection pattern each cycle to preform the required multiread functions as long a sufficient number of output lines are available. Thus it should be possible to simultaneously perform several pending evaluations of the l and d functions for different window sets, as long as the projections do not overlap. It is possible to check for such overlap by simply checking whether both the row and column dependent window set expressions are disjoint via the operation described in section 2.2.1.5.

The proposed parallel evaluation scheme addresses one of the principal problems associated with Lewin's algorithm. That is, as the search space window set w contracts, less and less of the hardware is usefully employed. The use of the parallel scheme allows us to employ a much greater portion of the hardware at any one time. An additional advantage arises from the fact that many small windows may be evaluated simultaneously. Given that Lewin's algorithm spends the bulk of its time evaluating small windows, for the simple reason that there are more small windows than large windows, such a capability may be used to greatly accelerate the execution of the algorithm.

The primary disadvantage of the parallel execution scheme is that a considerable amount of overhead may be required in order to schedule the order of evaluation for the windows involved and control the interconnection structure of the second stage. It may be possible to minimize the required overhead by selecting an order of evaluation which is oriented towards diagonal components, such as the Peano Order.

2.6.5. Generalization of the Folded Architecture.

Up to now we have considered the folded implementation of the multiread function in terms of a concrete geometrical configuration which arises from the structure of the Morton Order. We shall now generalize the concept of folding to take into account arbitrary order and dimension.

We shall begin by recognizing that the nature of the mapping between the input matrix I and the p vector is immaterial to the evaluation of the multiread, l,d and g(w) functions and consequently the execution of Lewin's algorithm and its variants. Thus if we were to specify the structure of interconnections for the two logical stages employed within the folded architecture, and then change the order of mapping of the I matrix while retaining the same order of interconnection with the p vector the results will be identical. Of course, the geometry of the interconnection structure will have been altered, which, in general, precludes of regular interconnection schemes. Of course, it is possible to implement the resulting interconnection pattern using a two stage HOPLA architecture. Such an implementation retains the limited fan-ins of the previous design while adding the flexibility of the HOPLA.

If we note that the variation in mapping order may simply consist of a rearrangement of bits, such as is the case for the raster scan order and the Morton order, we may conclude that the selection of row and column bits is arbitrary. Thus it is possible to designate any two arbitrary sets of bits as the row and column values and then proceed to formulate a valid folded interconnection scheme.

As we have noted previously, the multiread function and Lewin's algorithm are entirely independent of the dimensionality of the data employed. Consequently it should be possible to extend the principle of folding to higher dimensions. We may generalize the strategy of folding in the following manner: We first map the p vector into an a dimensional hypercube. We then select an axis and decompose the hypercube into a set of a-1 dimensional hypercubes each of which are normal to the axis selected. We then OR together the contents of each of these component hypercubes to obtain each of the elements of the projection vector for this axis. We then repeat the process for each of the a axes of the hypercube to obtain the a projection vectors required. We then evaluate the multiread function of each of these projection vectors and concatenate the results to obtain the multiread function for the entire set of data (the p vector).

As a concrete example, we consider the case of three dimensional folding. We begin with a p vector with 64 elements, which may be mapped to either a 8x8 square or a 4x4x4 cube as illustrated in figure 2.19. We may represent the contents of the cube by the use of cross sections as shown in the figure. If we employ the Morton order of mapping, the associated control masks consist of periodic planes of widths which are powers of two and are normal to each of the three axes. Thus, we may evaluate the multiread function by forming the projection vectors for the three axes of the cube and evaluating the multiread function of each projection vector. Each of the projection vectors may be obtained by the cube into its component cross sections along the desired axis and ORing the contents of each cross

section into a single logical value which, in turn, becomes an element of the the projection vector for that axis.

Just as in the case of two dimensions, we may interpret the planes that we employ in generating the multiread function as a set of primitives that may be employed to compose the desired control masks. Figure 2.19 illustrates the cross sections of three of these planes, each normal to a different axis. We may interpret the structure of these planes in cross section in the following manner. Two of the planes form single pixel wide periodic bar patterns in the x and y direction which may be used to compose control masks with periods smaller than that of the bar pattern. The third plane may be viewed as a single extended pixel that may be used to form the control masks that have periods greater than the bar patterns employed.

Unlike the two dimensional case, folding schemes which employ folding into higher dimensions cannot be implemented directly. The utility of such schemes is that they provide various interconnection patterns which may be implemented upon two stage HOPLAs. Each a-dimensional folding scheme is functionally identical, but employs a different number of active devices and a different level of fan-in. Thus the dimension of folding a provides a design parameter which allows for a tradeoff between the fan-in required and the number of active devices required. The fan-in and number of active devices required for each stage of a two stage a-dimensional folded implementation of the l and d function of a p vector of length 2^n:

First Stage:
Fan-In:
$$2^{(a-1)n/a}$$
Active Devices:
$$a2^{na+1}$$
Second Stage:
Fan-In:
$$2^{(n/a)-1}$$
Active Devices:
$$4n$$

For a value of n=20

a	First Fan-In	Stage Devices	Second Fan-In	Stage Devices
1	1	2097152	524288	80
2	1024	4096	512	80
4	32768	256	16	80
5	65536	160	8	80
10	262144	80	2	80
20	524288	80	1	80

As we may see form the table above, as we increase the value of a we increase the fan-in and decrease the number of active devices.

It is instructive to observe the results obtained for the extreme values of a, that is a=1 and a=n. In the case of a=1 the p and p' are simply mapped to themselves in the first stage, while the second stage simply consists of a non-folded single stage implementation of the multiread function. While in the case of a=n, the interconnection pattern of the first stage is identical to that of a single stage non-folded implementation of the multiread function and the second stage merely inverts the results of the first stage. Thus, surprisingly we may interpret our original implementation of the multiread function as either not folded or folded to a maximal extent!

2.7. Error Detection and Handling.

Up to this point it has been assumed that the execution of the algorithms that we have described proceed without error. Obviously, this is a false assumption, in that some degree of error occurs within any physical system. For this reason, it is important to determine the nature and impact of the presence of noise and error within our system and devise means for its detection and correction.
2.7.1. Error Quantification

The first task in addressing the issue of error is to to enumerate the various errors that may occur and to determine the severity of their impact upon the system. Obviously it is necessary to to determine which errors may occur before devising appropriate responses to them. But, it is equally important to determine the probability of occurrence of the error and its severity of impact upon the operation of the system. Such information is critical to determining the priority given to measures intended to prevent its occurrence. Additionally, information on the probability and severity of errors may be employed within the design of the system in order to minimize the overall severity of errors that may occur. Such a design tradeoff may actually increase the number of recoverable errors in order to minimize the number of severe or fatal errors.

2.7.2. Error Detection

There exist several methods for responding to errors that arise within a system. The first method, error detection, employs redundancies within data to detect the presence of corrupted data and prevent it from propagating through the system. The second method, error correction, employs redundancies within corrupted data to determine the most probable form of corruption so that it may be reversed. Finally, when errors are so extreme that they cannot be eliminated via correction or repetition, the third method, graceful degradation must be applied. Graceful degradation is a strategy which attempts to minimize the effects of error. Such a strategy either employs less efficient strategies that do not rely upon the corrupted data, or returns partial results.

Lewin's algorithm and our proposed variants are well suited to the use of all three methods. This arises from three features of the multiread function: The multiread function already contains a considerable amount of redundancy, which may be employed for the purposes of error correction. The multiread function operates upon flag addresses in such a manner that parity is, in some sense, preserved. This allows for the use of error correction codes. Finally, as we have noted previously, the multiread function employs self similar control masks with a sliding range of resolutions, which are employed to progressively converge upon a flag or a block of flags. Such an arrangement is well suited to graceful degradation schemes.

2.7.2.1. Error Detection Via Intrinsic Redundancy

The first line of defense against error is the intrinsic redundancy of the multiread function and the further redundancy of the l and d functions.

The first form of redundancy manifests itself in the binary representation of the window set expression. The binary representation of a window set consists of two binary vectors which are denoted by the superscripts 0 and 1 respectively. For a given bit position, if both vectors contain a false value (0) then that position contains a value of $\emptyset$. If, on the other hand, one of the bit positions contain a true value (1) then that position contains a constant value of either 1 or 0, depending upon whether the true value is within the 1 or 0 vector. But if both vectors contain a true value (1) within a given position, then an error has occurred. The sole exception to this situation is when all entries within both vectors contain true values, since this corresponds to the representation of the null set.

We may exploit the above redundancy to generate the following error test. A window set x is only valid if:

$$x^0 x^1 + (x^0 x^1)' = 0$$

Where 0 denotes a binary vector with all elements set to a false value. Such an error test may be integrated into the appropriate input latches of the LPU. If an error of this form is detected, the corrupted data must be discarded and error recovery invoked. The reason for this is that the $\underline{g}(w)$ function will map such a result to the null set.

Another form of redundancy relates to the relationship between the l and d functions and their relation to the search space w. As we have noted previously, the l and d functions are constrained to several specific outcomes during proper execution of our second algorithm. Either, at least one of the the functions must be equal to the search space, or, the results of the two functions must be a pair of disjoint window sets whose union is equal to that of the search space. Such redundancy may be used to trap errors by explicitly testing for these conditions. The two tests required are: Whenever l(w) and d(w) are evaluated the union of the two resulting window sets must be equal to the search space

window set w. When the results l(w) and d(w) are both proper subsets of the search space window set w the two results must be disjoint (i.e. have no intersection). These two tests are best implemented within the algorithm and performed by the LPU. Corrupt results detected by these two tests must be discarded since such errors imply incomplete coverage of the search space by the l and d functions.

2.7.2.2. Error Correction Coding

In addition to exploiting the inherent redundancies of of our system, it is desirable to incorporate additional redundancy for the explicit purpose of error detection and correction. Ideally such redundancy should provide an improvement in error handling while resulting in a minimal increase in the complexity of the the system.

For example, if we were to implement all of our interconnections in triplicate and selected results by majority vote, the resulting system would be very resilient to errors. Unfortunately, the resulting system would be more than three times as complex as the original. Under most circumstances such added complexity would be prohibitive.

Error correction coding schemes, on the other hand, may be employed to incorporate useful redundancies with minimal increases in complexity. Such coding schemes are most commonly used within data storage and compression, and are less commonly employed within computational tasks. The reason for this is that many arithmetic operations tend not to preserve the redundancies employed within the code. Error Correction Codes (ECCs) have been successfully used within electronic PLAs. We shall argue that ECCs are well suited for application to the multiread function and the worst case window set function (i.e. $g(w)$).

Consider the following error handling scheme employing l(w) and d(w) evaluation hardware. We first subdivide the input matrix into an array of equally sized disjoint cells each of which forms a window set. For each cell we designate a single valid pixel, the position of which is determined by the location of the cell that it is located within. We then mask all pixels which are not designated as valid within our evaluation hardware.

The masking scheme described above could be used for error detection and correction in several ways: A result which indicated a multiplicity of pixels within a single cell would obviously indicate an error, since only one valid pixel would exist per cell. A result that indicated a single pixel at a location other than a valid pixel location would indicate an error. The most probable correct result for either of these errors would correspond to the designated valid pixel within the cell under consideration. Since no two valid pixels could have adjacent addresses, any window set incorporating more than one variable bit would be the result of an error. Finally, in some circumstances, a window set that incorporated more than one cell would incorporate additional collapses that resulted from the correlation of of the addresses of the valid pixels enclosed, the absence of which would indicate an error.

Clearly, the scheme proposed above would be exceedingly wasteful, given that the bulk of the pixels within the hardware would be masked in such a scheme. But, it is possible to retain all of the advantages of the masked system within a practical scheme by retaining the same interconnection pattern, and discarding the masked pixels. The resulting scheme would be functionally identical to the masked configuration but would collapse each cell to the single valid pixel. Essentially such, a configuration would augment the address of each pixel with additional bits that could be used to validate the physical address. This is identical to the concept of error correction coding.

We shall employ the strategy of augmenting the address of each flag (pixel) to integrate error correction coding into the multiread and window set functions. For simplicity we shall employ one of the better known error correction codes, the Extended Hamming Code (EHC).

Given a consecutive sequence of $(m-\log_2(m)-1)$-bit binary numbers, Hamming code [50] appends $\log_2(m)$ code bits to each number in such a fashion as to ensure that any two numbers differ in at least two bit positions. Extended Hamming Code (EHC) appends one additional bit which ensures that any two numbers differ at least three bit positions. We may define the number of bit positions in which two numbers vary as their Hamming distance (for example the Hamming distance between 1010 and 0110 is equal to two).

Each of the check digits employed within the Hamming code are obtained by preforming an exclusive or (XOR) of a designated set of bit positions within the number to be coded. There exist several methods for designating which bits contribute to which check bit. The simplest method

consists of employing a binary count to designate the bits. In such a method, the binary numbers from 1 to 2^m-1 are listed in a column. Any number which is a power of two is then eliminated. For the remaining list, if the ith entry contains a value of one in the jth bit position, then the ith bit position of the number contributes to the jth check bit. For example we consider the case of m=8 where there are four data bits (a_3,a_2,a_1,a_0) and three check bits (c_2,c_1,c_0). We begin by listing the numbers from 1 to 8 eliminating powers of two (including 2^0=1) in a column:

3	0	1	1	a_0
5	1	0	1	a_1
6	1	1	0	a_2
7	1	1	1	a_3
	c_2	c_1	c_0	

The resulting formulas for the check bits are:

$c_0 = a_0$ XOR a_1 XOR a_3
$c_1 = a_0$ XOR a_2 XOR a_3
$c_2 = a_1$ XOR a_2 XOR a_3

The additional parity bit c_3 required to construct an extended Hamming code may be obtained by XORing both the data and check bits together:

$c_3 = c_2$ XOR c_1 XOR c_0 XOR a_3 XOR a_2 XOR a_1 XOR a_0

There are two attractive features to this method of generating Hamming codes. The first is that the same scheme may be extended to any number of data bits without reconfiguration by simply adding addtional check bits and extending the list of numbers. The second feature is that the higher order check bits are only influenced by the higher order data bits. This latter feature is particularly useful for checking errors within large windows and graceful degradation.

Other methods exist for generating Hamming codes, the most notable of these employing Linear Shift Registers (LSRs). The advantage of such methods generally lies in simpler physical implementation and positional separation of data and check bits. Unfortunately such methods do not have the two advantages that have been cited for the method noted in the previous paragraph.

Once the formula for generating the check bits has been determined, implementation is straightforward. All that is necessary to do is to generate the check bits for each address and then append these bits to the least significant portion of the address. The resulting augmented address may then be used to form an augmented S matrix, which may in turn be used to construct the control masks for the new configuration. A more intuitive method for obtaining the control masks associated with the check bits is to XOR together the control masks associated with the address bits that contribute to the check bit formula. The form of the multiread, l,d and q functions are identical with the exception that the augmented S matrix is employed.

All of our proposed algorithms will operate without modification employing functions augmented with check bits. Even without modification, our intrinsic error correction checks shall be augmented by the presence of redundant bits. Of course, it is necessary to incorporate some error checking and correction procedures within our system to derive the full advantage from the incorporation of Hamming code into our system. Such procedures are applied to the results of the l and d functions for each cycle. These procedures differ from traditional Hamming code checks in that they must be able to handle the presence of variable bits (Ø) within window set results. Two rules determine the handling of such variable bits. The first rule is that if any data bits contributing to a given check bit are variable, it is not possible to verify the state of the check bit. The second rule is that if one and only one (data or check) bit of any check bit formula is variable, then at least one of the bits is in error.

The an error detection scheme incorporating the two rules noted above requires two phases. The first phase implements an independent conventional check of the two binary vectors that compose the window set expression (i.e. l^0 and l^1 or d^0 and d^1). such a check consists of reconstructing each check bit from the received data bits and comparing the result with the received check bit. Both checks will validate constant bits due to the fact that the complement of a valid Hamming code is also a valid Hamming code. Such a check detects errors when the bits involved are constant and also detects

errors resulting from the presence of a single variable bit. The second phase of the checking process detects the presence of multiple variable bits within each check equation and suppresses error signals generated by the first phase of error correction. This phase implements the first rule cited above. By employing the combination of these two operations, it is possible to extract the maximal amount of error detection information from window sets.

It is possible, given a sufficient amount of error detection information, to engage in error correction. Error correction is possible when the hamming distance of the constant bits exceeds the number of errors detected. Such a curcumstance is most likely when the window set is quite small (on the order of one to four flags in size). Since such error correction schemes   .

The implementation of the two phase error detection scheme is straightforward. The operations required may be implemented within the LCU in the form of discrete logic associated with the input latches.

The implementation of the first phase consists of connecting each check bit and its associated data bits to a common XOR gate. A true (1) result indicates an error condition associated with the corresponding check bit. Such check circuitry should be independently implemented upon the two binary vectors (0,1) which compose a window set representation.

The implementation of the second phase is somewhat more complex. The first step in the implementation is to perform a bitwise NOR operation upon the two component binary vectors of the window set register of interest. The resulting binary vector contains a true value (1) within each bit position which contains a variable bit ($\emptyset$) within the window set expression. The second step in the implementation is to route each check bit and its associated data bits to a circuit which returns a false value if more than one of its inputs is true. The resulting output is false when a the presence of variable bits renders error detection employing the given check bit invalid. The final step of the second stage is to use the results of the circuits noted above to suppress error signals generated by the first phase when such signals are invalid. This is achieved by individually ANDing the result of the second stage for each check bit with the two signals for the same bit generated by the first phase.

The final step within the error detection process is to OR together the various error detection lines and feed the result to the process control logic within the LCU. When this line generates a true output then the algorithm must initiate error handling procedures.

Obviously, error correction codes other than Hamming codes may be employed within our application. Berger, m-of-n and Fire codes are well suited to use within PLA architectures, and may prove superior to Hamming code in the long run.

2.7.3. Error Handling

When an error occurs it is necessary to discard the corrupt data and obtain correct data before proceeding with the execution of the algorithm. Valid data may be obtained by either repeating the process that obtained the data or by attempting to correct the corrupted data. If it is impossible to obtain valid data by either means, it is then necessary to proceed without such data in a manner that minimizes the effect of the missing data. Such a strategy is known as graceful degradation.

2.7.3.1. Error Correction Through Repetition.

The simplest means of recovering from random errors is to discard the corrupt data entirely and repeat the evaluation process. Such a strategy is effective as long as the error rate is sufficiently low that repeated errors are rare. As the error rate increases, the number of repetitions necessary to obtain a correct result increases as well. At a certain error rate, there will be a strong probability that a corrupted result will defeat the error detection system employed, resulting in an undetected error occuring. Thus, it is necessary to limit the number of repetitions allowed to a small number.

2.7.3.2. Error Correction Through Coding

Another method of correcting corrupted data is to employ the error correction codes that we have incorporated within our functions. Given a sufficient number of valid check bits (those which have not been suppressed by the second phase of our error detection scheme) it is possible to determine the most probable correct result from the corrupted data. As with error detection, a sufficiently high bit error rate will overwhelm such error correction schemes allowing undetected errors to propagate

through the system. For this reason it is inadvisable to employ such methods when bit error rates exceed a prescribed value.

Error correction is possible when the set of valid check and data bits forms a set of correct codes with a hamming distance greater than the number of detected errors by one. Correction is achieved by replacing the corrupted code with the correct code which is the shortest distance from it. The precise method for determining this corrected code is dependent upon the means by which the original code has been generated. In the case where all check bits are valid, Hamming code is capable of correcting one bit error and extended Hamming code is capable of correcting up to two bit errors.

The primary utility of error correction within the current design is to augment the reliability of single flag (or pixel) results. Such a capability is particularly useful when the error encountered is systematic, arising from an effect such as crosstalk and thus not amenable to correction through repetition.

2.7.3.3. Graceful Degradation

When it is possible to detect an error but it is not possible to eliminate it through correction or repetition it is necessary to proceed with the execution of the algorithm without the data in question. The strategy of graceful degradation seeks to minimize the impact of such errors by the incorporation of appropriate error handling procedures. The first line of assault for such strategies is to proceed with less efficient algorithms that do not require the erroneous data. If such measures fail, then a strategy of containing the effects of such failures so as to allow for the continued execution of the algorithm. In such a strategy, the portion of the result obscured by the error is labeled as corrupt and the best aproximation to the result is given. Our proposed algorithms are well suited to the incorporation of graceful degradation strategies.

The first strategy for employing graceful degradation within our algorithms is to ignore erroneous collapses. Given that collapses of the search space only accelerate the execution of our algorithm, the failure of a collapse to occur merely results in a reduction in the efficiency of the algorithm. In the case of our two proposed algorithms there is also a commensurate decrease in the efficiency of compression due to the fact that smaller blocks are employed to encode flags.

In the case of the second algorithm such a strategy may take an interesting form. If either the l or d function goes into a state of persistent failure, the the algorithm will rely solely upon the other function for a source of collapses. Such an event amounts to the degradation of our second algorithm to either our first algorithm or Lewin's original algorithm depending upon whether the failing function may still generate null set results. Such a form of persistent failure might conceivably result from an asymmetric implementation of such functions.

In the case of situations where error rates are sufficently high as to defeat error detection or indicative of systematic error it is necessary to terminate evaluation of the portion of the input under consideration. In the face of such a termination it is the role of graceful degradation strategies to contain the effects of such an error. Such containment of failure is easily achieved within our algorithms due to the recursive nature of both the algorithms and the functions employed.

When errors that preclude further evaluation are encountered within our algorithms, the appropriate response is to record the current search space in the form of it's window set and designate it as indeterminate and terminate recursion. This allows the algorithm to proceed with the next search space while indicating the location and extent of the error encountered.

The impact of such indeterminate data depends upon the nature of the application in which it is employed. In the case of ordered retrieval, it is necessary to locate the missing flags by alternative means. Such a task is simplified by the fact that the location of the missing flags is constrained by the window set associated with the indeterminate value. In the case of image decomposition and compression, on the other hand, this amounts to the conversion of a lossless decomposition to a lossy decomposition. The presence of indeterminate data corresponds to a loss of detail within portions of the image. Within limits, such a loss of information may be tolerable within imaging applications.

If, under a given set of circumstances, the bit error rates for the evaluation of our functions beyond a given resolution are prohibitive, then it is possible to operate our system at a lower resolution without hardware modification. The only modifications required consist of adjusting the thresholds of the active devices upward to compensate for the brighter logic signals, and disabling the least signifigant bits of the functions, which are associated with the lower resolutions. In this fashion we may

dynamically reconfigure the structure of our system in such a manner as to provide some level of functionality under arbitrarily hostile conditions.

2.8. Conclusion

We have described the multiread function and its application within Lewin's algorithm. We have proposed extensions upon Lewin's algorithm that accelerate the task of ordered retrieval as well a providing the potential for compression. We have shown that these algorithms allow for the efficent parallel quadtree decomposition of images as well as several other practical applications. We have shown that these algorithms inherently display global parallelism and are well suited to implementation upon our proposed HOPLA architecture. Finally, we have proposed a means of "folding" the multiread function, a strategy which allows us to limit the fan-in required for the system while allowing for extended parallelism.

Chapter 3

Experimental Overview

3.1.Introduction.

Within this chapter we shall out..ine our program of research. We shall describe our objectives, and review the state of current research.

The basic expremental objectiv of the current project is to evaluate the practical limits of global architectures, and specifically the HOPLA architecture. In order to achieve this objective, various configurations of HOPLAs shall be constructed and evaluated under various conditions. A special emphasis shall be given to the relationship between the performance of the system under low levels of optical power.

In order to reach these objectives, four major tasks must be preformed: A POHM printer must be constructed in order to generate the N^4 interconnect required for the computer. The N^4 interconnects of various configurations must must be reconstructed, evaluated, and calibrated to ensure proper operation of the intended computer. Proper basic operation of the computer must be verified and, the preformance of various configuations of the computer must be compared. Finally the operation of the computer must be evaluated under declining levels of optical power. Due to the nature of these tasks, it is anticipated that this cycle will need to be repeated several times to obtain our desired results.

At the present time an initial version of the POHM printer has been constructed, and initial results are being collected and evaluated. The current status of this work shall be described followed by a description of the current direction of research. This shall be followed by a rough schedule of research which is yet to be done. Finally, the anticipated results of the research and their implications for the future shall be discussed.

3.2. Status of Current Research:

A this time a holographic laboratory has been established at the USAF Photonics laboratory Griffiss AFB. An initial POHM printer has been constructed and has shown fully automatic operation. Initial POHMs have been made and are under initial evaluation. A high contrast Spatial Light Modulator (SLM) is available and shall be introduced to into the system shortly, allowing for initial computer operation and evaluation.

3.2.1 The POHM Printer:

In order to generate a Holographic Optical Programmable Logic Array, it is first necessary to construct a means of generating an arbitrary N^4 interconnect of the appropriate dimensions. A Page Oriented Holographic Memory printer serves both as the means of generating an N^4 interconnect, as well as the framework of the interconnect itself.

The operation of the N^4 interconnect which we shall employ within this system has been described within previous chapters. A sche.natic diagram of the the interconnect is given in figure 1.5.

As we have noted previously, the N^4 inteconnect employs an array of adjacent Fourier transform holograms each of which contain t'ie image of a different control mask. Each image projects back upon

the input SLM. When these holograms are simultaneously illuminated by a reconstruction beam conjugate to that of the original reference beam, they project all of the control masks upon the input SLM. It is the function of the POHM printer to generate this array of holograms.

Within our experiment, the POHM printer, and the N^4 interconnect which implements the HOPLA employ the same optical system. This configuration has been selected for two reasons: It allows us to minimize the duplication of scarce and expensive resources, and it greatly simplifies the problem of alignment. Given this configuration, once a successfully operating POHM printer has been constructed, it is relatively simple to convert it to an associated HOPLA system.

In order to generate the required holograms, the POHM printer must automatically record a series of Fourier transform holograms. Each of these holograms differ in content and optical configuration. The POHM printer must thus be both capable of controlling the various aspects of hologram exposure, and of changing its optical configuration for each hologram recorded.

The exposure of each separate hologram within the POHM differs in three respects; The control mask is different, the location of the hologram is different, and the origin of the object beam is at a different location. Each of these changes must be addressed by the POHM printer in a different manner. Changing the control mask simply consists of changing the content of the SLM within the system. The location and extent of each hologram is controlled by an aperture which prevents the hologram from being exposed over the entire plate. The location of each hologram is changed by shifting the location of this aperture. Finally, in order to change the location of the point source that forms the origin of the object beam, it is necessary to employ an optical fiber to generate the point source. Moving the point of origin of the object beam then simply a matter of moving the position of the output end of the optical fiber.

3.1.2.1. Design Issues.

At the present time, a POHM printer employing fixed transparencies has been sucessfully constructed at Rome Laboratory and is undergoing preliminary analysis. An electronically addressed, high contrast version of a 256x256 SIGHT-MOD Magneto-Optic SLM is currently being integrated into the system to provide for fully automated operation. We shall now review the design and construction of this POHM printer. Throughout this discussion we shall consider the various design tradeoffs involved.

We shall divide our description of our POHM printer into three major divisions, the issues related to the optical design of the system, the mechanical design of the system and the recording of the holograms.

3.1.2.1.1. Optical Design Issues.

In the previous chapter the basic constraints upon N^4 interconnect systems, and consequently, POHM systems implementing such interconnects, were considered. Within this section we shall consider issues related to the practical optical design of the POHM printer.

Figure 3.1 illustrates the optical layout of the POHM printer. The recording geometry of the system is a variant of the quasi-Fourier-Franhofer hologram. The reference and object beams employ independent optical trains. The system employs a collimated reference beam in order to simplify conjugate reconstruction. In contrast, the optics object beam tolerates a considerable degree of aberration, since the conjugate reconstruction employed in the interconnect should cancel out such aberrations[2]. Another notable feature of the optical system is the use of optical fibers for beam routing.

The system employs Spectra Physics 2025 water cooled argon ion laser emitting a single line linearly polarized beam at a wavelength of 514nm. The laser incorporates an oven heated etalon and a Littrow mirror to achieve operation in a single longitudinal mode. The laser employs current feedback as opposed to power feedback since it has been observed that the former mode of operation minimized pointing variation while maintaining relatively good power stability.

Tuning of the laser is complicated by the lack of a scanning Fabry-Perot interferometer for the observation of the output spectra. Tuning is achieved by maximizing both the intensity and visibility of the fringes generated by the output of an unbalanced Michelson interferometer. The presence of mode hopping is detected by long term monitoring of the interferometer fringe pattern. The coherence length of the laser is verified by observing the visibility of a deep scene transmission hologram of a

retroreflective measuring bar. Measurments obtained in this fashion indicate a nominal coherence length of roughly one meter. This indicates the need for more accurate tuning of the laser to obtain true single mode operation.

The laser employed must be mechanically isolated from the rest of the optical system, since it vibrates considerably. This is achieved by mounting the laser on a table separate from the remainder of the optical system and routing the beam to the system by the use of a single mode polarization preserving optical fiber. Figure 3.2 illustrates this arrangement. The laser beam first exits the laser, passes through an electro-mechanical shutter, then passes through a half-wave retarder plate, and finally enters a fiber coupler which injects the beam into the optical fiber. The shutter is located on the laser table, since it is also a considerable source of vibration. The half wave plate aligns the polarization of the beam to the birefringent axis of the optical fiber. At the output end of the fiber the beam is decoupled and collimated and then passed through a polarizing beamsplitter in order to ensure linear polarization.

The output beam generated by the above arrangement is subject to power fluctuations arising from several sources. The primary source of the fluctuations originate within the laser, both directly from the variation of output power, and indirectly due to variations in beam pointing resulting in fluctuations in fiber coupling efficiency. Additional variations in coupling efficiency result from various fiber optic effects, such as internal reflection[51], cladding modes, and residual birefringence. At best it is possible to minimize the various sources if fluctuations. The graph in figure 3.3 indicates the variations in power encountered in normal operation. In order to compensate for these fluctuations it is necessary to monitor the power fed into the main system. This is achieved by diverting a small portion of the beam entering the system into a monitoring detector as shown in figure 3.1.

The main optical system consists of two separate paths, the reference beam path and the object beam path. The actual division of the incoming beam into the two paths is achieved by employing a polarization type variable beamsplitter. This allows us to continuously vary the critical reference beam-object beam ratio.

The layout of the reference beam path is quite simple. The beam is first directed through a detour in its path in order to ensure the matching of the optical path length with that of the object beam path. The beam then passes through a diffraction limited collimator. The beam is expanded to a diameter of 50 centimeters. The resulting collimated beam then impinges upon the holographic plate at an angle of 30° to the normal. This angle was selected since a beam incident at a sharper angle would intersect the mounting hardware of the object beam optical system.

Many Fourier transform hologram recording schemes employ a reference beam which originates as a point source at the plane at which the transparency is located. The advantage of such a scheme is that the reference and object beams share a common optical system and the incident angle of the reference beam may be greatly reduced. Such a scheme is not employed in the current system for two reasons. The first problem with such a scheme would be the difficulty of mounting the reference beam fiber within the SLM housing. The second problem is that the reference beam generated would inevitably posses a measure of aberration. Such an aberrant reference beam would greatly complicate reconstruction of the resulting hologram, especially when a conjugate reconstruction beam is employed as is presently the case.

The configuration of the object beam path is considerably more complex than that of the reference beam. Thus we shall divide our discussion of this path into two sections, the fiber optical portion, and the conventional optical system. It is necessary to employ fiber optics in order to be able to shift the focal point of the object beam for each exposure.

We shall first discuss the fiber optic portion of the object beam path. The beam is first coupled into an optical fiber in a manner identical to that of the the arrangement used to transfer the laser output to the main optical system. As in the previous arrangement, a half wave plate is employed to ensure that the polarization of the beam is aligned parallel to the birefringent axis of the optical fiber. At the output end of the fiber the beam exits directly from the fiber, the end of the fiber effectively serving as a pinhole type spatial filter with a numerical aperture of 0.1. The actual emission from the end of the fiber approximates the form of an expanding elliptical gaussian beam. This is a result of the anisotropic structure of the fiber core necessary to maintain polarization. It is the tip of the output end of the fiber which forms the origin of the object beam for the conventional optical system.

The remainder of the object beam path consists of a conventional optical system. Figure 3.5 presents a detail of this system. This system is identical to the N^4 interconnect except that the propagation of the beams involved are conjugate to those of the interconnect. The function of the system is to first collimate the expanding beam emitted from the optical fiber, then illuminate the SLM or transparency containing a desired control mask, and finally focus the resulting object beam upon the holographic plate. The system achieves this by the use of three lenses, each of them a plano-convex lens of the same focal length, which is, in this case 150 mm. The first lens, located directly after the end of the optical fiber, operates as a field lens. It serves to steer the beam towards the center of the second lens, as well as slightly increase the divergence of the beam. The field lens is necessary to prevent vingetting within the system. The second lens serves to collimate the expanding beam. The collimated beam then propagates through the SLM and its associated polarization analysers. The third lens then focuses the resulting beam onto the holographic plate. The focal plane of the system is displaced from the holographic plate by a short distance (roughly 8mm), in order to achieve a more uniform distribution of energy at the recording plane.

Although it would be optimal to place the second and third lenses in the optical system in near contact with the SLM or transparency within the system, this is not possible due to the physical configuration of the SLM employed.

The object beam optical system described above is subject to considerable abberation, especially in off axis operation. As we have noted previously, these aberrations should be cancelled out by the conjugate reconstruction employed within the N^4 interconnect.

The object and reference beam paths recombine at the holographic film plate. An aperture is located immediately in front of the film plate to prevent each hologram from exposing the entire plate. Scatter from this aperture must be minimized since it contributes noise to the hologram being recorded. The aperture currently employed consists of a thin aluminum shim painted with flat black paint. The edge of the hole is stripped of paint in order to eliminate irregular scatter from individual paint particles. The aperture is nearly in contact with the plate so as to ensure the greatest overlap between the object and reference beams.

3.1.2.1.2. Mechanical Design Issues

We now consider the mechanical issues associated with the POHM printer. As we have noted previously, both the location of the object beam focal point (which consists of an optical fiber tip) and the plate aperture shift for each of the many holograms recorded within a POHM. It is therefore necessary, between exposures, to move both the fiber tip and the aperture to the appropriate locations for the next exposure. This is achieved by the use of an automated micropositioning system. This system is integrated with the general control system which controls and monitors the other acive components of the system, such as the laser, the shutter and the SLM. Both the object beam fiber tip and the plate aperture are mounted on micropositioner systems. Both of these systems consist of Kilnger linear x-y positioners arranged to operate along a plane normal to that of the optical axis of the object beam optical system. All of the micropositioners are of the stepper motor type, and have a nominal resolution of one micron. Each of the four micropositioners is under the independent control of the controller computer.

The automated exposure system controls the motion of the micropositioners during an exposure sequence. Previous to each exposure, both the fiber tip and the aperture are moved to their prescribed locations for that exposure. The system then waits for a set period of time. This delay allows for the vibrations induced by the motion of the micropositioners to damp out. The system then exposes the hologram, and then moves the fiber tip and the aperture to the next prescribed position.

3.1.2.1.3. Holographic Design Issues.

Within this section we shall conssider the design issues which are associated with the process of generating the holograms recorded by the POHM. These issues not only concern the fabrication of the hologram itself, but also concern the control of the environment to ensure optimal recording of the holograms.

The first issue we shall consider is the control of the environment. In the recording of any hologram, a variety of environmental factors such as vibration and air currents strongly influence the quality of the exposure. Such factors are random in their effects and, unless carefully controlled, may serve as the predominant influence on the uniformity and diffraction efficiency of the hologram involved. Such factors must be minimized by the proper design of the recording system.

In order to minimize the effects of air currents the entire main optical system is surrounded by a felt curtain. This curtian serves several purposes. It blocks air currents and it absorbs sounds within and without the system. Finally, it serves to absorb scattered light within the system.

In order to minimize the effects of vibration the entire system is mounted on a vibration isolated table. All components within the system are both damped and braced in such a manner as to minimize vibration.

As noted previously, the laser employed within the system has been mounted on a separate table so as to mechanically isolate it from the remainder of the system. This not only eliminates a major source of vibration within the system, but also a major source of heat and thus convection currents.

It is possible to monitor the overall stability of this holographic system by the use of real time holographic interferometry. In this technique a hologram is recorded using the system. This hologram is then processed and returned to precisely the same location that it was recorded in. When such a hologram is then illuminated by both the original reference beam and object beam, interference occurs between the holographic reconstruction of the object beam and the object beam itself. By observing the resulting fringes it is possible to determine the stability of the optical system.

By employing the above method with the current system it is possible to make a qualitative determination of the stability of the system. Little or no short term variation in the fringes generated in this manner has been observed under normal conditions, implying sufficient stability for the recording of high quality holograms.

As we have noted in previous sections, the power delivered to the optical system is subject to considerable variation over time. Since exposure energy is one of the most critical factors associated with the generation of holograms. it is necessary to compensate for such power variations. This is achieved by employing an exposure control system which samples the power at regular intervals during the exposure and integrates the results to determine the appropriate exposure time.

Exposure control is mediated by the main control computer. The computer is directly connected to the shutter system allowing it to control the exposure time based upon the information arriving from the monitoring system. The power of the main system is monitored by the use of a photodetector which detects light split from the main beam by a weak beamsplitter as indicated in figure 3.1. The signal from the photodetector is first amplified and then digitized by an analog to digital converter integrated into the control computer. During an exposure, the control program regularly samples the signal and numerically integrates the results in order to determine the total exposure energy at that time. Once the exposure energy has reached the desired value the control computer closes the shutter ending the exposure.

The next issue that we shall consider is the recording medium and processing regimen employed for the hologram. The current system employs conventional silver halide emulsions as the recording medium. Once the holograms are recorded, they are processed with a tanning developer, a rehalogenating bleach, and a weak colloidal developer to obtain a phase hologram.

This combination was selected for several reasons: The process results in highly repeatable results. The sensitivity of the medium is very high . The resulting holograms display a high degree of stability and are not subject to enviromental variations. The hologram recorded experiences minimal distortion of its reconstructed wavefront as a result of processing. Such advantages are obtained at the cost of a lower diffraction efficency than would be obtained from other media such as dichromated gelatin, or photopolymers.

Dichromated gelatin, and photopolymer emulsions were rejected on the basis of the relatively poor repeatability and uniformity of processing, vulnerability to humidity and temperature, poor sensitivity, and and the tendency of the medium to distort as a result of processing.

The specific emulsion employed in recording is AFGA 8E75 HD AHI Millimask plate film. This is a silver halide and gelatin emulsion with a emulsion thickness of 6 μm and an average grain size of 0.044 μm. The sensitivity of the emulsion is 200 μJ/cm^2[52] at 514 nm and the resolution is greater than 3000 lines/mm. The emulsion incorporates an antihalation dye which attenuates internal reflection, this
is augmented by a coating of #33 metal stripping paint applied to the back surface of the glass substrate. The dimension of the plate employed is 63.5 mm square. No pretreatment is used.

After exposure the holographic plate is processed in a three step process, the plate is first developed with a tanning developer and then bleached with a rehalogenating bleach and finally re-developed with a weak citric acid developer. The final result of this processing is a high efficiency hologram of the phase type. The process achieves this result by first developing the plate, converting the exposed silver bromide crystals into filamentary or 'black' silver in the conventional manner. The rehalogenating bleach then converts the filamentary silver back into silver bromide crystals. The resulting distribution of the silver bromide crystals is dependent upon the distribution of the filamentary silver, which in turn corresponds to the exposure pattern. The irregular distribution of these silver bromide crystals causes a modulation of the index of refraction forming a phase hologram. The plate is then immersed in a weak solution of citric acid, and illuminated with a bright light. The citric acid solution acts as a weak developer, converting the silver halide grains into colliodial or 'brown' silver. This technique is known as redevelopment, and although it results in some attenuation, it does improve the stability of the resulting hologram and reduce the degree of scatter. The resulting hologram is then rinsed in desensitizing dye and dried.

There are several important features to this processing regimen. The first feature is that the resulting hologram is a phase modulated volume hologram, and therefore capable of high diffraction efficiency. The second feature is that unlike processes that employ fixes, or solvent bleaches, the chemical content of the emulsion remains relatively constant. In fact, if the redevelopment process is eliminated, the processed hologram is identical in composition to an unexposed plate. By minimizing the variation in volume and content of the hologram during processing, the distortion of the emulsion, and thus the recorded fringe pattern is minimized. This leads to minimal distortion of the holographic image during reconstruction. The third notable feature of the processing regimen is that the phase modulation generated by the hologram is a result of the migration of the silver bromide crystals. Such grain migration is limited in scope, resulting in little or no response to low spatial frequencies. In essence the processing operates as a high pass spatial filter. Since the holographic fringe pattern itself is of a high spatial frequency, the high pass filtering essentially operates as a low frequency noise filter.

We now describe the specific processing regimen employed (chemical formulas will be given in Appendix B): After exposure, the holographic plate is developed for 2 minutes in CWC2 developer. Optimal results are obtained when development results in an optical density of 2. The plate is then rinsed in flowing water for 4 minutes. The plate is then bleached for 2 minutes in a Copper Sulfate bleach. This is followed by another 4 minute rinse in flowing water. The plate is then placed in the redeveloper solution and exposed to intense white light for 4 minutes. The plate again rinsed for 4 minutes in flowing water. The plate is then soaked for 4 minutes in a combined solution of Kodak indicator stop bath and Kodak PhotoFlo 200. This step binds the indicator dye of the stop bath to the remaining silver bromide grains desensitizing them, while the PhotoFlo 200 reduces the surface tension of the solution to facilitate drying. All processing is done at a temperature of 74°F. Finally the plate is then removed, and left to dry in still air.

Several variations of this processing regimen shall be explored in the future. The use of varying development times shall be examined, as well as the use of PBUQ bleach. In theory[53] PBUQ bleach results in less distortion, and results in holograms with greater long term stability. A comparison of the merits of using redeveloper shall also be examined.

3.1.2.2. Reconstruction, Evaluation and Testing.

Having described the design, construction and evaluation of the POHM printer we now consider the reconstruction and evaluation of the holograms generated by the system. In addition we shall also consider the use of the POHM printer to evaluate the performance of the recording media and processing techniques.

Once a POHM has been generated, it must be reconstructed and evaluated. The reconstruction of the component holograms of the POHM involves the use of a reconstruction beam conjugate to that of the original reference beam. By employing this conjugate reconstruction the image of each control mask reconstructs at the same location within the optical system as the original mask was located. By reconstructing the hologram in this fashion, it is possible to evaluate the resolution, contrast, uniformity, efficency and registration of the hologram. Within our system, the reconstruction of the POHM may be achieved by the use of the POHM printer. The POHM is placed back into the same location as it was recorded at and then illuminated with a reconstruction beam conjugate to that of the reference beam, as per figure 3.6. The holograms then reconstruct their control mask images at

the plane of the SLM or transparency. The plate aperture may then be positioned to obscure all but one of the holograms.

There are several methods for observing the resulting image. The simplest method is to replace the SLM with a film plate in the same location and to expose the image upon it. The exposed plate may then be developed and fixed to obtain a photographic print of the reconstructed image. This image may then be analyzed by microscopic examination or by the use of micro-densitometer scans. Another method is to image the reconstructed image upon a CCD camera. The resulting image may be continuously adjusted and monitored and may then be stored by the use of a frame grabber. These advantages are gained at the cost of resolution, as well as a susceptibility to aliasing between the control mask and the CCD pixels.

As well as allowing us to examine the the quality of reconstruction of individual control masks, the methods noted above also allow us to simulate the generation of the window set function $g(w)$. As we have noted in chapter 2, the same POHM that implements the multiread function for a given order may be employed to generate the corresponding window set function. By simultaneously reconstructing the appropriate set of control masks it is possible to generate the logical complement of any desired window set at the SLM plane. Thus by employing the appropriate masking of the POHM and recording the resulting pattern at the SLM plane it is possible to evaluate the suitability of the POHM for this task. Unfortunately, for the lack of an appropriate OSLM, the window set function shall not be explicitly implemented.

In the the same vein, it is also possible to evaluate the registration of the control masks by simultaneously recording either identical or complementary control masks. In the case of identical mask reconstruction the resulting image should be identical to the reconstruction of a single hologram. In the case of complementary control masks, the resulting image should correspond to a control mask with every pixel activated. In both cases a shift in the relative reconstruction position of the two masks should result in a combined image that should clearly indicate the nature of the lost registration. Such a method should detect both net translations and distortions of the two masks. The advantage of this method is that it is unnecessary to register the projected image relative to the position of the original mask location within the system.

Simple POHMs which have been generated by the use of the POHM printer, have been evaluated by the use of the above methods. Film recordings of the single and multiple hologram reconstructions have been successfully recorded upon the holographic plate film. The resulting images have not yet been subjected to intensive analysis.

It is also possible to allow the image to propagate through the remainder of the optical system allowing it to concentrate the light. This corresponds to operating a single channel of the N^4 interconnect. If the SLM is absent the concentrated light gives us the practical efficiency of the hologram. If we exchange the SLM for a moving aperture, such as a knife edge, we may compare the resulting variations in the intensity pattern against those predicted by theory. For example, the bar patterns associated with the Morton order control masks should present a very distinctive pattern as they are obscured by a knife edge which is parallel to the bar pattern as per figure 3.7. Variations from this pattern (adjusted for the diffraction effects of knife edge) may be interpreted to obtain information upon the overall uniformity, resolution, and registration of the mask. In the same manner, a square aperture may be used to evaluate the response of the control masks to a "pixel" of a given resolution. The advantage of the above methods is their simplicity, in that they only employ a single detector and simple aperture.

Using the initial POHMs recorded by the printer, the above methods have been used to measure the relative diffraction efficencies of the component holograms. The results of these measurements shall be discussed.

Finally we shall consider the use of the POHM printer for sensitometery testing. The testing and evaluation of various processing techniques under varying exposures is greatly facilitated by the use of the POHM printer. The automated exposure system allows us to generate a large number of holograms on the same plate, each with a different exposure energy. Furthermore by modifying the configuration of the printer it is also possible to generate arrays of identical diffraction gratings and arrays of uniformly exposed areas. Three configurations of the printer shall be considered.

The first testing configuration blocks the object beam entirely and only exposes the reference beam. Each exposure is made with the aperture at a different location, and with a different exposure energy. The resulting plate consists of a series of positions that have been exposed to a uniform beam of light

with a known exposure energy. The resulting plate when developed and fixed may be used to relate exposure and density.

The second testing configuration employs the printer with the SLM and the third lens removed as per figure 3.8. Each exposure is made with the object beam origin centered at the optical axis and the plate aperture located at a different position. As in the case of the first test configuration each exposure is made with a different exposure energy. In this case both the reference beam and the object beam consist of stationary collimated beams. The resulting array of holograms, when processed, all consist of identical diffraction gratings which have been recorded under differing exposures. These gratings may be evaluated and used to relate the nominal diffraction efficency.

The third testing configuration simply consists of the POHM printer in its normal configuration. By recording an array of identical holograms with identical exposures we may determine the effects of the varying recording geometries upon the diffraction efficency of the resulting holograms. By recording an array of identical holograms with differing exposures, it is possible to roughly evaluate the influence of exposure upon the practical diffraction efficiency.

Obviously, all of the above testing configurations generate results which are subject to several forms of systematic error which are dependent upon the position of each sample. These variations result from the nonuniformity of the reference and object beams, and in the third configuration, variations in the recording geometry. Thus it is always necessary to generate test plates with exposures of equal intensity. Such plates may be used to determine the nature of such variations and, when possible, calibrate for such effects. Even with calibration, some systematic error will persist. If the presence of such errors is taken into account, the resulting data may be employed to optimize the recording process.

Preliminary measurements relating density, relative diffraction efficiency and exposure have been measured by the use of these techniques. Figure 3.9 displays the D-LogE curve for Millimask film under the our standard processing regimen. Figure 3.10 displays the relative diffraction efficiency versus exposure, for a set of holographic lenses. These initial results are highly suspect, since problems involving coherence length and processing temperature had not been resolved.

3.3. Planned Research

Having described the status of our research so far, we shall now describe the work which remains to be done. We may divide the remaining work into two major tasks. We must construct an N^4 interconnect and then we must evaluate its performance under a range of conditions. In order to achieve the latter task it is necessary to devise appropriate test procedures which shall allow us to rapidly evaluate the reliability of a given interconnect.

3.3.1. Construction of the N^4 Interconnect

Having generated a set of POHMs containing the appropriate control masks it is necessary to construct the corresponding N^4 interconnect. Such an interconnect essentially corresponds to the operation of the POHM printer under time reversal as per figure 3.6. A reconstruction beam conjugate to the reference beam simultaneously reconstructs the holograms that compose the POHM. The conjugate images generated by these holograms propagate "backwards" through the object beam optical system, passing through the SLM and concentrating at the origin of the object beam for each hologram. Given that the optical layout of the interconnect is, with the exception of the conjugate reconstruction beam, identical to that of the POHM printer, it is possible to employ the same system both as a printer and an interconnect. Within the current project we shall such a scheme, employing a combination printer/interconnect assembly. In order to allow operation as an interconnect, two additions to the POHM printer must be made. A reconstruction beam path must be added and a detector system must be located at the output plane.

3.3.1.1. The Reconstruction Beam System.

The reconstruction beam path generates a collimated beam propagating in the opposite direction of the reference beam. This is achieved by diverting the reference beam from its usual path and routing it to a second collimator which generates the reconstruction beam as per figure 3.11.

It is possible to align the reconstruction beam by allowing it to propogate into the reference beam collimator. If the reconstruction beam is properly aligned, it should pass through the reference beam collimator pinhole and the resulting thin beam exiting the collimator should follow the same path as

that of the reference beam. Such an alignment procedure constrains both the transverse position (two axes) and the direction of propagation (two angular axes) of the reconstruction beam.

Of course, the alignment of the hologram is equally important as the alignment of the reconstruction beam. In order to ensure that the processed hologram is located in the same position as when it was exposed, the printer interconnect employs a special plate holder, originally designed for holographic interferometry[54], which employs a system of three rods and three spheres to register the hologram in a unique position. The accuracy of relocation is on the order of several microns. It is possible to verify the accuracy of relocation by simultaneously illuminating one of the holograms with both its original reference and object beam. The interference of the reconstructed object beam and the original object beam result in a projected fringe pattern. The structure of this fringe pattern may be analysed to determine the relative accuracy of the reconstruction.

The addition of the reconstruction beam path to the existing POHM printer is the only optical modification necessary to allow operation of the printer as a N^4 interconnect. When operated as an interconnect, both the reference beam path and the object beam path are disabled and all power is diverted to the reconstruction beam path. In this mode of operation, the SLM is located at the input plane of the interconnect and functions as the input device. The output plane of the interconnect is located at the same plane as the origin of the object beam. In order to sense this output it is necessary to locate a detector system at this location, the implementation of which is the topic of the next section.

3.3.1.2. The Detector System

As we have noted, the output of the N^4 interconnect consists of an image located at the same plane as the origin of the object beam within the POHM printer. Such an image must be detected and and processed in order for it to be useful. Within an optical computer such a function would ideally be performed by an active optical, or optoelectronic device. Such a device would detect and process the output signal, and generate an optical signal which is dependent upon the processed signal. In the case of the HOPLA the processing involved takes the form of a threshold and invert operation which results in a discrete logical signal. In the case of outputs that contain a small number of outputs, such as the multiread function, it is possible to detect the output employing an array of photodetectors, and perform the analysis and processing with electronics.

The primary objective of the current project is to evaluate the operation of various configurations of a single stage HOPLA. For this reason the nature of the signal at the detector plane is more important than the processed signal. Therefore, the output signal is detected electronically, digitzed, and then processed by the control computer. The advantage of this technique is that it collects the maximal amount of data on the state of the system, while allowing for the maximum amount of flexibility in the analysis of the data.

We shall now describe several possible arrangements of a detector system and compare the merits and disadvantages of each scheme. There exist three major schemes for implementing the detector system: We may employ an array of discrete detectors, one for each output channel. We may image the entire output plane upon a detector array and analyse the resulting image. Finally we may employ a single detector which moves to each output channel in turn.

The most obvious arrangement for the detector system is to employ an array of discrete detectors, one for each output channel. The advantages of this scheme are that it would provide for rapid operation of the detector system and that it would most closely resemble a practical implementation of the HOPLA architecture. The drawbacks of such a scheme are its complexity and inflexibility. Each detector would require an associated amplifier and either a thresholding element and a latch, or if analog results were required, an associated sample and hold circuit. In addition, in order to collect analog data, it would be necessary for the Analog to Digital Converter to sequentially address the sample and hold circuit for each channel.

The complexity of the detector array scheme is further complicated by the need to calibrate the individual detectors, support circuitry and threshold mechanism. Such a task is simplified within a production system in that the threshold may be adjusted to compensate for variations in the sensitivity and linearity of the dectector support circuitry. In comparison, within an experimental system it is necessary that each detector and circuit is calibrated to an identical sensitivity, so as to allow for the accurate measurement of the incoming signal. For the same reason it is necessary to be able to adjust the threshold mechanism so as to provide for differing thresholds for each position during the development of the system.

Obviously, within a fixed design intended for production, few of the disadvantages cited apply. A fixed configuration detector array with integrated thresholding circuitry would be ideal for production systems. Irregularities of production could be compensated by electronically adjusting the thresholds of each detector.

A more flexible means of detecting the output of the HOPLA is to image the output plane of the system by employing a detector array with a resolution much higher than each output location. The resulting image may then be calibrated and analyzed within software in order to determine the output signal.

The primary advantages of such a strategy are its ease of implementation, flexibility, and diagnostic capabilities. Such a scheme may be implemented with a combination of an "off the shelf" CCD camera with integrated readout electronics and a sufficently powerful control computer. Since the regions associated with each output bit are defined within software, they may be arbitrarily rearranged by modifying the analysis program. Finally since such a scheme returns an image of the output plane, it may be used to analyze the effects of aberration, diffraction and crosstalk upon the system. Additionally, such as strategy may be implemented by using any imaging detector system including; Charge Injection Devices (CIDs), intensified CCDs, and Photon Imaging Detectors (PIDs). Such a range of different detectors allow for a range of sensitivities and speeds sufficient for most experimental work.

The basic disadvantage of such systems is that the amount of computation necessary for calibration and analysis exceeds that of the system that implements it. Although such a flaw is fatal within production systems, it has little impact upon the use of such a scheme within an experimental system. The disadvantages encountered within the use of this strategy within experimental systems are the increase in noise associated with the use of such arrays, and the added complexity of the optical system. Due to the complications of addressing such detector arrays and the presence of dark current noise with each pixel of the array, such a detector scheme is likely to be subject to greater noise than a system employing a single detector. In addition, the size of most detector arrays employed within imaging are smaller than the optimal size of the output plane of the HOPLA. By reducing the dimensions of the output plane, the effects of crosstalk arising from diffraction effects are increased.

Within the current project, imaging detector arrays shall be employed for detection within relatively high light levels.

The final option for a detector system is to employ a single optimized detector mounted upon a micropositioner stage which shuttles between the location of each output bit. Such a strategy operates so slowly that it is only applicable for use within experimental systems. The advantages of such a scheme are simplicity, flexibility and accuracy. A single well calibrated detector may be mounted upon the same micropositioner which is used to move the focal point of the object beam during the recording of the POHM. The route that the micropositioner takes is identical to that of the object beam focal point during recording. The sensitive area of such a detector may be modified by exchanging the aperture of the detector. Such a strategy is clearly as flexible as that of the recording system. There are several advantages to employing the same detector for each measurement. The properties of the detector employed may be rigorously determined and such properties shall apply to each measurement. In addition, a far more complicated detector may be employed than could be contemplated for a detector array. The dimensions of such a detector may even exceed that of the spacing of the output bit positions, a configuration physically impossible within a detector array strategy.

Within the current system, shuttling detector systems shall be employed for precise measurements of output levels and for low photon count measurements. Pin-diode detectors are currently being integrated into the system for use with moderate light levels and a photon counting photomultiplier tube and support circuitry are on order.

3.4. Outline of Future Research

At this time, the bulk of the ground work associated with the construction of a working HOPLA architecture has been achieved. Within this section we shall describe the work that remains to be done in order to implement an operational HOPLA architecture and evaluate its performance.

3.4.1. The POHM Printer.

At the present time the POHM printer has shown successful automated operation with fixed transparencies. The remaining task is to integrate the SLM into the system and evaluate the preformance of the resulting system. The evalauation of the 256x256 SIGHT-MOD magneto-optic is nearly finished and integration into the system is anticipated shortly. The automated exposure software is presently being modified to allow the image upon the SLM to be modified previous to each exposure.

The schedule for the integration of the SLM is as follows. The SLM and a set of film polarizers shall be integrated into the optical system. As set of POHMs shall be recorded with identical control masks but differing levels of exposure. The resulting holograms shall be evaluated in comparison with identical exposures performed with fixed transparencies of the same control mask. The resulting holograms shall be compared with a POHM employing the same exposure for each facet. Each facet shall then be reconstructed at the input plane and evaluated for resolution, contrast, and efficiency and the presence of vingetting within the system. The film polarizers shall then be replaced with Glan-Thompson prisms and the process repeated.

Once the optimal configuration and exposure of the system has been achieved, a POHM incorporating varying control masks shall be recorded at a constant level of exposure. Once this has been achieved, each facet of the resulting hologram shall be analyzed for resolution and contrast. If necessary, variations in efficiency and contrast resulting from variations in regularity and spatial frequency shall be compensated for by changes in exposure and reconfiguration of the control masks. Once uniformity has been achieved, POHMs containing control masks for the multiread function shall be recorded. Several POHMs each with a different mapping order shall be recorded.

3.4.2. Operation and Evaluation of the HOPLA.

Once a Set of POHMs containing the required control masks have been generated and evaluated, it is possible to construct and evaluate a HOPLA which implements the multiread function. As noted previously, by simply disabling the reference and object beams within the POHM printer and introducing a reconstruction beam conjugate to that of the reference beam the desired HOPLA may be formed.

Once the HOPLA has been constructed it is then necessary to evaluate the performance of the unit. In order to do this it is necessary to evaluate the response of the unit to various inputs displayed upon the SLM and evaluate the relative intensity of signals noise and crosstalk. Once reliable measurements of these quantities have been taken it is possible to determine the optimal configuration of the control masks and set the associated thresholds for each of the output bits. It is then necessary to use the HOPLA to evaluate the multiread function for various applications and determine the reliability of the system. Finally it is necessary to determine the lowest light level at which operation is possible.

The appropriate testing regimen for the HOPLA consists of the following steps for each output bit. The output should first be measured with the SLM all light and all dark. These two tests measure the maximum and minimum light levels encountered for each output. The next test is to measure the output bit with the SLM set to the complement of the control mask. Such a pattern should result in the maximum intensity anticipated for a false signal since the effects of input and output crosstalk are maximized in this case. The next test is to measure the results of a progression of window sets which converge from the entire control mask to a single pixel. Such a test measures the range of intensities associated with a true value ranging from the maximum value to the possible value. This regimen should be repeated both with all facets of the POHM illuminated and with a each single facet illuminated. This latter test may be achived by the employing the movable aperture used during recording within the POHM printer.

At this stage within the evaluation process, it is possible to empirically set the threshold values for each output bit. After this has been done, the next phase of the regimen is to evaluate the response of the HOPLA to various inputs. Given that each pixel constitutes an input to the system, it is impossible to evaluate the response of the system to every possible combination of inputs. Therefore it is necessary to determine a series of input test patterns that serve as representative samples of the input anticipated.

We propose that a set of test patterns may be obtained by the following method: Form a set of masks from the following three sets; the control masks employed within the multiread function implemented, the control masks from a differing mapping of the multiread function, the control masks generated by

a mapping determined by the output of a Maximal-length Linear Shift Register (MLSR). The latter masks obtained by employing an S matrix generated by the output of a MLSR, which to all statistical tests appears to be random[50]. A valid test pattern may then be obtained by randomly ANDing a given number of control masks from this set (with care taken not to use complementary masks). Each group of masks provides a differing feature to the test pattern. The presence of control masks of the same order of mapping as the multiread function ensures that collapses will occur, ensuring that the output pattern contains information. The presence of control masks with a mapping other than that of the multiread function supply regular patterns that do not necessarily result in collapses. Finally, the presence of MLSR generated control masks supply random patterns. The results of each test pattern may first be determined by the use of simulation, and then verified by evaluation on the HOPLA.

The next stage of evaluation is to run a large series of random test test patterns of the type described above and evaluate the reliability of the HOPLA involved. The three measures of reliability to be evaluated are, the absolute error rate, the error detection rate, and the portion of detected errors that may be corrected by the various means previously cited.

The final stage of the evaluation process consists of employing the HOPLA to evaluate the multiread functions for our various algorithms. Lacking the means of directly generating the $l(w)$ and $d(w)$ functions it is possible to simulate their operations by the use of the appropriately masked and inverted portions of the input matrix I. The input for these algorithms may include both images and randomly distributed flags of varying densities. The error rate encountered within this phase of the regimen shall most likely be lower than that of the prevous stage. The reason for this is that the recursive nature of the algorithms that we employ progressively eliminate from consideration various portions of the functions employed for the length of each recursive descent.

3.4.2.1. Evaluation of the HOPLA at Low Light Levels

Once the operation of the HOPLA has been evaluated at moderate light levels by the methods described within the previous section it is then necessary to evaluate the operation of the HOPLA under decreasing light levels. Hypothetically, the sole absolute limit of reliable operation of the HOPLA is the shot noise limit. If the contrast of the SLM and POHM components are sufficiently (on the order of 1000:1) reliable operation may be achieved with less than a thousand photons per true signal. Given that each output bit represents the result of 32,768 logical operations (assuming a 256x256 input SLM) an HOPLA operating at these light levels would perform more than one logical operation per photon propogating through the system. While counterintuitive, such computation is physically possible. This is a result of the fact that the probability distribution at the output plane of the HOPLA is proportional to the intensity pattern generated by a classical electromagnetic wave propagating through the system [6]. Caulfield [6] has dubbed such operations as Wave Particle Duality (WPD) computation and argues that they may operate at lower energy limits than conventional computation. Successful operation of of the HOPLA at such levels should validate such arguments.

The first stage in the evaluation of the HOPLA system at low light levels is to enclose the system in an "dark box" a light tight, baffled enclosure designed to minimize the presence of background light arising from sources within the laboratory and from scatter within the system. In addition, a system of baffles [55] should be incorporated within the system. The light for the reconstruction beam may be introduced into the system via a shielded fiber optic link.

Once the system has been appropriately enclosed, the next step is to employ the appropriate shuttling detector to measure the output of the system at the output plane. At the intermediate level, between the moderate light levels employed within the previous section and the photon counting regime, a paired pin-diode detector system, which electronically divides the signal from the output and a signal split off form the light source[56], may be used. Within the photon counting regimen, a Photo-Multiplier Tube (PMT) system may be employed. When employing the PMT system the output of the laser should be chopped in order to minimize the effects of noise. Furthermore, the system should employ the reciprocal counting strategy. Within such a strategy, the quantity measured is the amount of time necessary to gather a fixed count. The advantage of such a strategy is that since the number of counts measured is constant, the associated shot noise is equally constant. Such measurements may be thresholded by comparing the time period measured, with the amount of time necessary to integrate a proportional amount of energy from the light source.

Once the system has been reconfigured for low light level operation, we may repeat the testing regimen described within the previous section. Since the speed of measurement of the system shall be

limited by the use of a shuttling detector, an emphasis should be placed upon the use of the random test patterns of the previous section. The reason for this is that such random patterns cover the largest possible range of operation with the least cycles.

Finally, when reliable operation is no longer possible at a given light level it is possible to degrade the system to a simpler, lower resolution system by the methods described in section 2.7. By employing such a strategy it is possible to determine the minimum light level for reliable operation of a range of system of varying complexity.

3.5. Conclusion

Within this chapter we have described our experimental work to date on the implementation of a single stage massively interconnected HOPLA implementing the multiread function. We have also described the schedule of research for the completion of this project.

References

1. H.J. Caulfield, "Parallel N^4 Weighted Optical Interconnections" Appl. Opt., Vol. 26, pp. 4039- 4040, (1987)

2. J.Shamir, H.J. Caulfield and R.B. Johnson, "Massive Holographic Interconnections and their Limitations", Appl. Opt., Vol. 28, pp 311-324, (1989)

3. H.E. Elion and V.N. Morozov Optoelectronic Switching Systems in Telecommunications and Computers, Marcel Dekker, N.Y. (1984)

4. P.S. Guilfoyle and W.J. Wiley, "Combinational Logic Based Digital Optical Computing Architectures" Appl. Opt., Vol. 27. pp. 1661-1673 (1988)

5. P.S. Guilfoyle "Optical Digital Computer Design Specification Phase I Final Report" Unpublished (1988)

6. H.J. Caulfield and J. Shamir, "Wave Particle Duality Considerations in Optical Computing", Appl. Opt., Vol. 28, pp. 2184-2186, (1989)

7. H.J. Caulfield and J. Shamir, "Wave Particle Duality Processors-Characteristics, Requirements and Applications" (Accepted for publication in Appl. Opt.)

8. H.J. Caufield, J. Shamir, J.E. Ludman and P. Gregus, "Reversibility and Energetics in Optical Computing" (Accepted for publication in Optics Letters)

9. W.T. Cathey and W.J. Miceli, "Digital Computing with Optics", Proc. of the IEEE, Vol.77, p. 1558-1572, (1989)

10. M.M. Mirsalehi, M.A.G. Abushagur and H.J. Caufield, "Optical and Optoelectronic Computing" in Advances in Computers, Volume 28 Academic Press (1989)

11. J. Tanida and Y. Ichioka, "Modular Components for an Optical Array Logic System" Appl. Opt., Vol. 26, pp 3954-3960,(1987)

12. A. Huang "Parallel Algorithms for Optical Digital Computers" Tech. Digest, IEEE Tenth Int. Opt. Comput. Conf.,pp 13-17

13. A.P. Goutzoulis, E.C. Malarkey, D.K. Davies, J.C. Bradley, and P.R. Beaudet, "Optical Processing with Residue LED/LD Lookup Tables", Appl. Opt., Vol. 27, pp. 1674-1681

14. R.A.Athale, "Optical Matrix Processors", SPIE Proc. Vol. 634 Optical and Hybrid Computing, pp. 96-111 (1986)

15. M.A. Harrison, Introduction to Switching and Automata Theory, McGraw-Hill, N.Y., (1965)

16. R.E. Miller, Switching Theory, Volume I: Combinational Circuits John Wiley and Sons, N.Y., (1965)

17. S.A. Kupiec and H.J. Caulfield, "Digital Optical Wave Particle Duality Computers:A Practical Example, (Submitted to Applied Optics)

18. M.M. Mirsalehi and T.K. Gaylord, "Logical Minimization of Multilevel Coded Functions" Appl. Opt., Vol 25, pp. 3078-3088

19. S.L. Hurst, D.M. Miller and J.C. Munzio, Spectral Techniques in Digital Logic, Academic Press, London, (1985)

20. R.J. Lechner and A. Moezzi "Synthesis of Encoded PLAs" in Spectral Techniques and Fault Detection M.G. Karpovsky, Ed., Academic Press, Orlando, (1985)

21. M.G. Karpovsky, Finite Orthogonal Series in the Design of Digital Devices, Halsted Press, New York, (1976)

22. E. Fredkin and T. Toffoli, "Conservative Logic", Int. J. Theor. Phys. Vol. 21, pp 219-253, (1982)

23. L. Szillard, "Uber die Entropieverminderung in einen Thermodynamischen System bei Eingriffen Intellegenter Wesen." Z. Phys. Vol. 52, pp. 840-856,(1929)

24. L. Brillouin, Science and Information Theory, Académic Press, Boston, (1956)

25. R. Landauer, "Irreversibility and Heat Generation in the Computing process", IBM J. Res. Vol. 5, pp 183-191, (1961).

26. M.H. Lewin, "Retrieval of Ordered Lists From a Content Addressed Memory.", RCA Rev. Vol. 23, pp 215-299, (1962)

27. A. Wolinsky, "A Simple Proof of Lewin's Ordered Retreival Theorem for Associatve Memories", Communications of the ACM Vol.1., pp 488-490, (1968)

28. M. Murdocca and B. Sugla, "Design for an Optical Random Access Memory", Applied Optics Vol. 28, pp 182-188, (1989)

29. E.H. Frei and J. Goldberg, "A Method for Resolving Multiple Responses in a Parallel Search File.", IRE Trans. Vol. EC-10, pp 718-722, (1961)

30. B.Parhami, "Associative Memories and Processors: An Overview and Selected Bibliography", Proc. IEEE Vol. 61, pp 722-730, (1973)

31. T. Kohonen, Content Addresable Memories, Second Edition, Springer-Verlag, Berlin, (1987)

32. A.Yariv, S-K. Kwong and K. Kyuma, "Demonstration of an all optical associative Holographic Memory", Appl. Phys. Let. Vol 48, pp 1114-1116, (1986).

33. N.H. Farhat, D. Psaltis, A. Par ita, "Optical Implementation of the Hopfeld Model", Applied Optics Vol 24, pp 1469-1475, (1985)

34. D. Psaltis, H. Farahat, "Optical Information Processing Based on an Associative Memory Model of Neural Nets with Thresholding and Feedback", Optics Letters Vol 10, pp 98-100, (1985)

35. P.B. Berra, K. Brenner, V. T. Cathey, H.J. Caulfield, S.H. Lee and H. Szu, "Optical Database/Knowledgebase Machines", Applied Optics Vol. 29, pp. 195-205, (1990)

36. C. Warde and J.Kottas, "Hybrid Optical Inference Machines: Architectural Considerations", Applied Optics Vol. 25, pp. 940-947, (1986)

37. A.R. Butz, "Space Filling Curves and Mathematical Programming", Information and Control Vol. 12, pp. 314-330, (1968)

38. A.R. Butz, "Convergence with Hilberts Space Filling Curves", J. of Computer and Systems Sciences Vol. 3, pp. 128-146, (1969

39. T. Bially, "Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction", IEEE Trans. Inf. Theory Vol IT-15, pp 658-664, (1969)

40. R.J. Stevens, A.F. Lehar and F.H. Preston, "Manipulation and Presentation of Multidimensional Image Data Using the Peano Scan , IEEE Trans. on Pat. Anal. and Mach. Intel. Vol. PAMI-5, pp. 520-526, (1983)

41. E. Kanin and E. Walach, "A Fractal Based Approach to Image Compression", IEEE Trans. on ASSP, (1986)

42. H. Samet, "The Design and Analysis of Spatial Data Structures", Addison-Wesley, (1990)

43. H. Samet, M. Tamminen, "Computing Geometric Properties of Images Represented by Linear Quadtrees", IEEE Trans. on Pat. Anal. and Mach. Intel. Vol. PAMI-7, pp. 229-240, (1985)
44. G.M. Hunter and K. Steglitz, "Operations on Images Using Quad Trees", IEEE Trans. on Pat. Anal. and Mach. Intel. Vol. PAMI-1, pp. 145-153, (1979)

45. E. Kawaguchi, T. Endo and J. Matsunaga, "Depth-First Picture Expression Viewed from Digital Picture Processing", IEEE Trans. on Pat. Anal. and Mach. Intel Vol. PAMI-5, pp. 373-384, (1983)

46. E. Kawaguchi, T. Endo and J. Matsunaga, "On a Method of Binary-Picture Representation and Its Application to Data Compression." IEEE Trans. on Pat. Anal. and Mach. Intel. Vol. PAMI-2, pp. 27-35, (1980)

47. R.J. Collier, C.B. Buchardt and L.H. Lin, Optical Holography, Academic Press, Orlando, (1971)

48. B.B. Mandelbrot, Fractals Form, Chance, and Dimension, W.H. Freeman and Co., San Fransisco, (1977)

49. N.H. Farhat, Advances in Holography, Volume 3, Marcel Dekker, New York, (1976)

50. W.W. Peterson and E.J. Weldon Jr., Error Correction Codes, Second edition, MIT Press, Cambridge, Ma., (1972)

51. W.J. Tomlinson, A. Sharhar, and R.J. Deri, "Use and Misuse of End Facet Reflections in the Characterization of Optical Waveguide Directional Couplers", Applied Optics Vol. 30, pp 2961-2969, (1991)

52. J.W. Gladden, R.D. Leighty, "Recording Media" in Handbook of Optical Holography, H.J. Caulfield, ed., Academic Press, New York, (1979)

53. H.I. Belkhagen, N.J. Phillips and W. Ce, "Chemical Symmetry-Developers that Look Like Bleach Agents For Holography.", SPIE San-Jose Conference on Practical Holography, (1991).

54. N. Abramson, The Making and Evaluation of Holograms, Academic Press, London, (1981)

55. C.L. Wyatt, Electro-Optical System Design For Information Processing, Mc Graw Hill, New York, (1991)

56. P.C.D. Hobbs, "Reaching the Shot Noise Limit for $10", Optics and Photonics News Vol.2, pp. 17-23

Table of Figures

Appendix A

The table below displays the pertinent information on the compression of the attached figures. Both the DF bintree and the DF quadtree compressions employed the grey coded Peano order.

DF Bintree

Image	cycles	collapses	Compression
Jessica	4850	2679	3.69:1
RADC	5522	3199	3.11:1
Test	1042	651	16.86:1
UAH	3086	1892	5.20:1

DF Quadtree

Image	cycles	collapses	Compression
Jessica	4266	2157	5.18:1
RADC	5273	2638	4.24:1
Test	1092	367	22.15:1
UAH	3211	1363	6.84:1

Appendix B: Chemical Formulary
CWC2 Developer

Part A:

20 g Catechol
10 g Ascorbic Acid
10 g Sodium Sulfite
50 g Urea
1 l Distilled Water

Part B:

60 g Sodium Carbonate
1 l Distilled Water

A working solution is formed by mixing equal quantities of A and B before use.

Source: D.J. Cooke and A.A. Ward, "Reflection-Hologram Processing for High Efficiency in Silver-Halide Emulsions", Applied Optics Vol. 23, p 973, (1984)

Copper Sulfate Bleach

35 g Copper Sulfate
10 ml Acetic Acid
110 g Potassium Bromide
1 l Distilled Water

Source: J. Blythe, "A Novel Approach To Colour Processing", Wavefront Vol. 2, No. 3, p.23, (1987)

Re-Developer #2

10 g Ascorbic Acid
1 l Distilled Water

Dilute with water 40:1 for working solution.

Source: N. Phillips, "Bridging the Gap Between Soviet and Western Holograpy", Handout at Lake Forest College Holography Workshop II, July 1990

```
x1|x2|x3||zi|z2|z3
0 |0 |0 ||1 |0 |1
0 |0 |1 ||0 |1 |1
0 |1 |0 ||0 |1 |1
0 |1 |1 ||0 |1 |0
1 |0 |0 ||1 |0 |0
1 |0 |1 ||1 |0 |1
1 |1 |0 ||0 |1 |1
1 |1 |1 ||1 |0 |1
```

Figure 1.1

```
Terms |x1 |x'1|x2 |x'2|x3 |x'3| |z1|z2|z3|
------+---+---+---+---+---+---+-+--+--+--+
x'1x'2
x'1x3
x1x3
x2x'3
x'2x'3
```

Figure 1.2

Inverter Element

SLM

Detector Array

SLM

LED Array

Figure 1.3

SLM

SLM
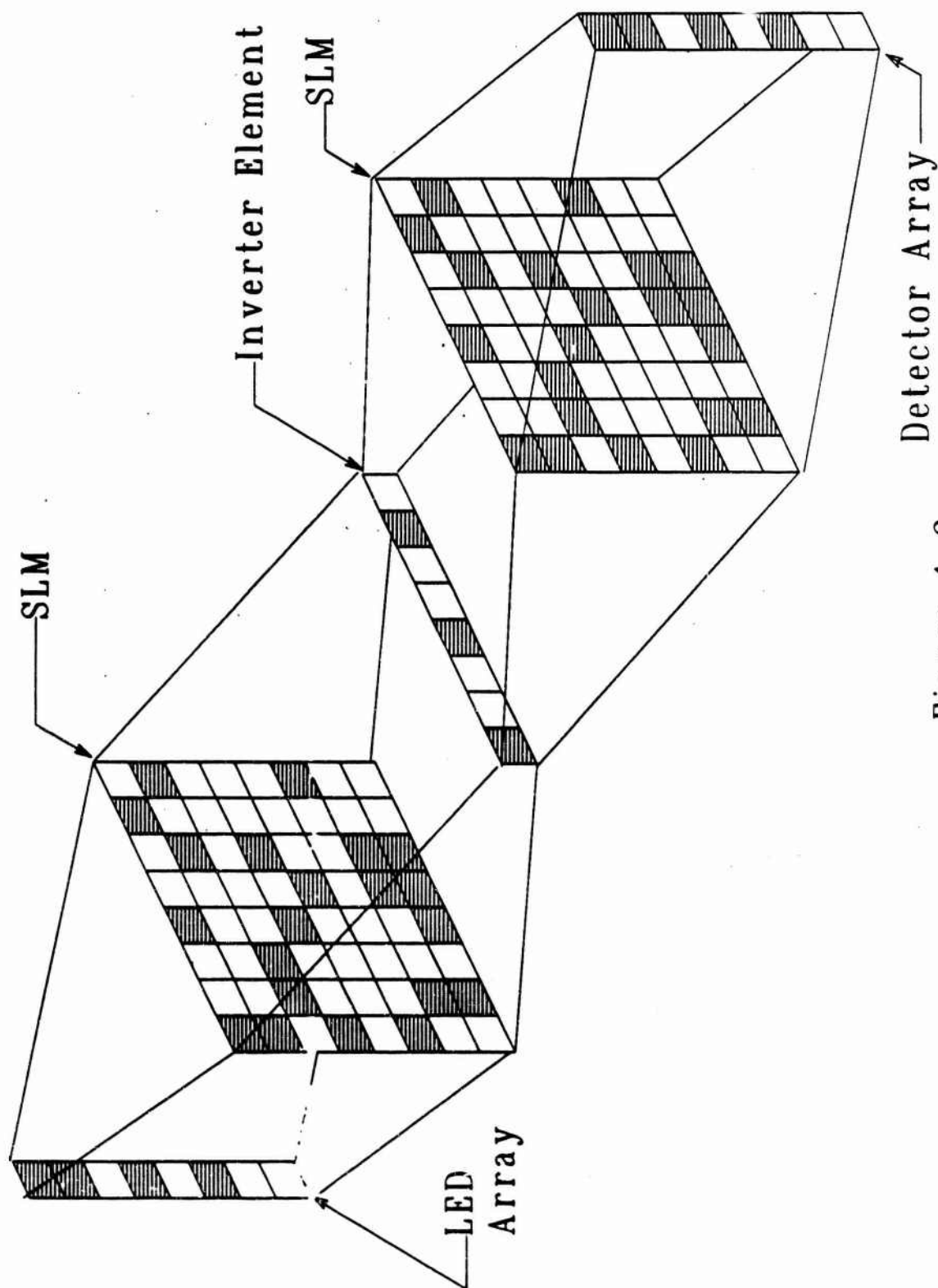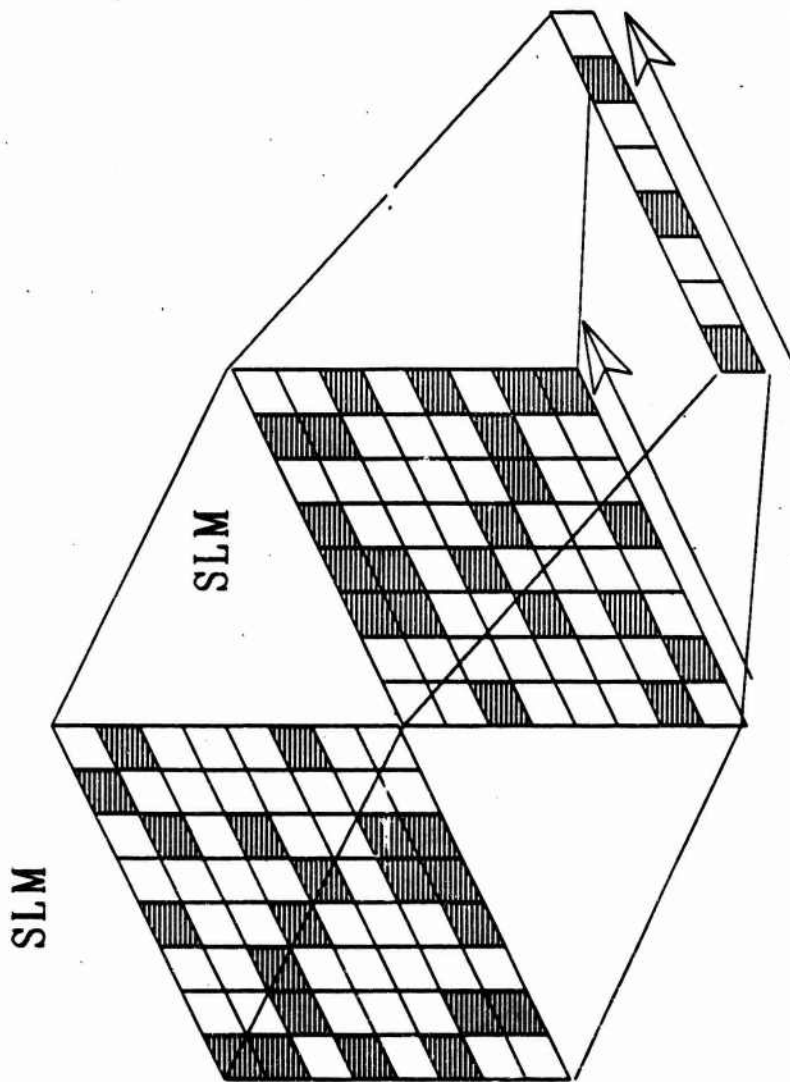
Shift and Invert
Detector Array

Figure 1.4
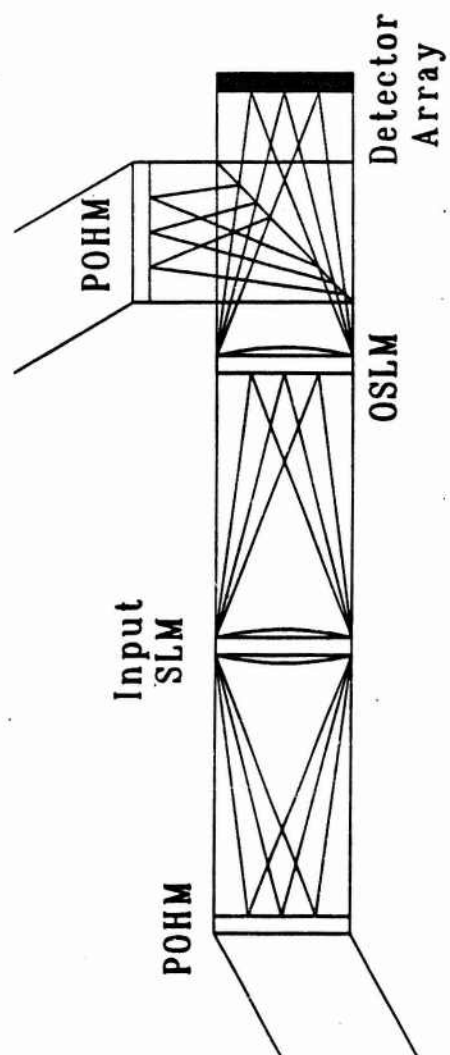
Figure 1.5
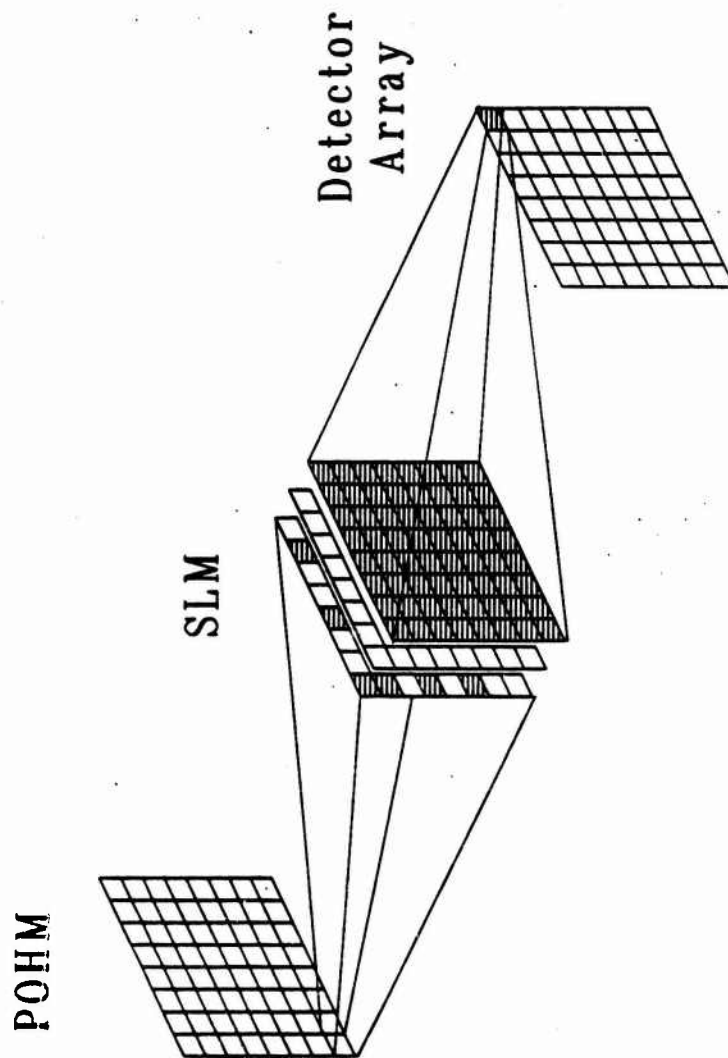
POHM

SLM

Detector Array

Figure 1.6

Figure 1.7

Figure 2.1
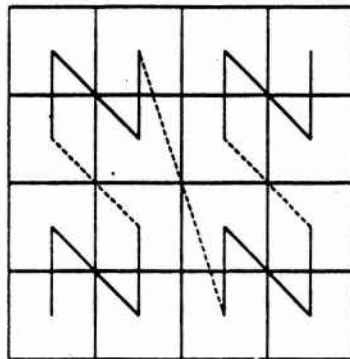
Peano
Order

Morton
Order

Raster Scan
Order

Figure 2.2

Figure 2.3

l collapse

no collapse
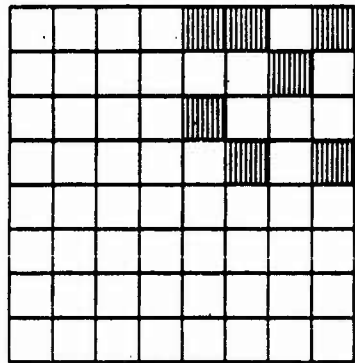
d collapse
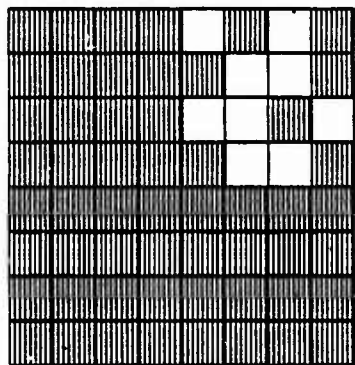
l is null set
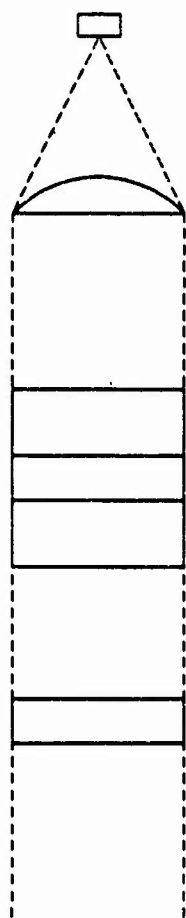
l and d collapse
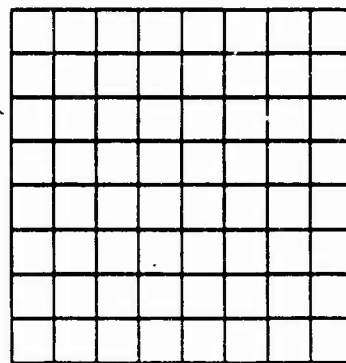
d is null set
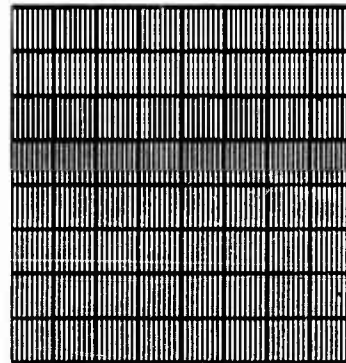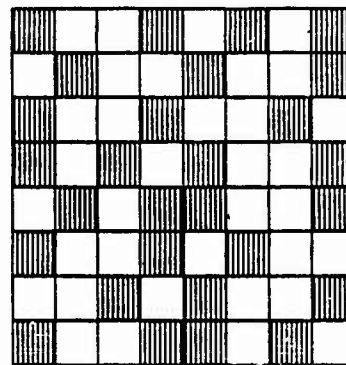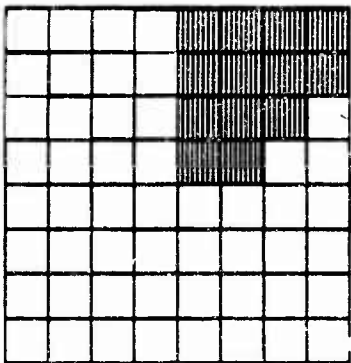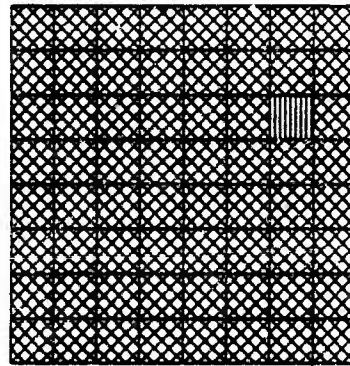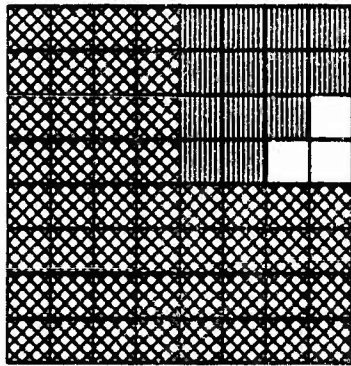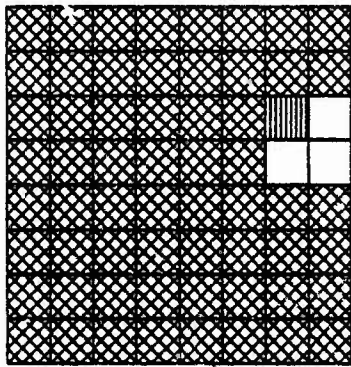
Figure 2.4

Figure 2.5
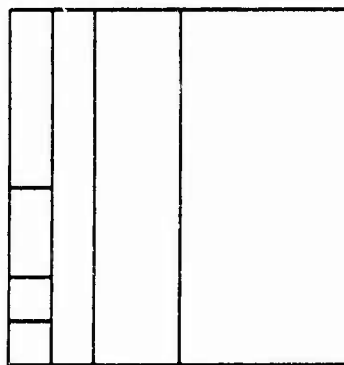
Raster Scan
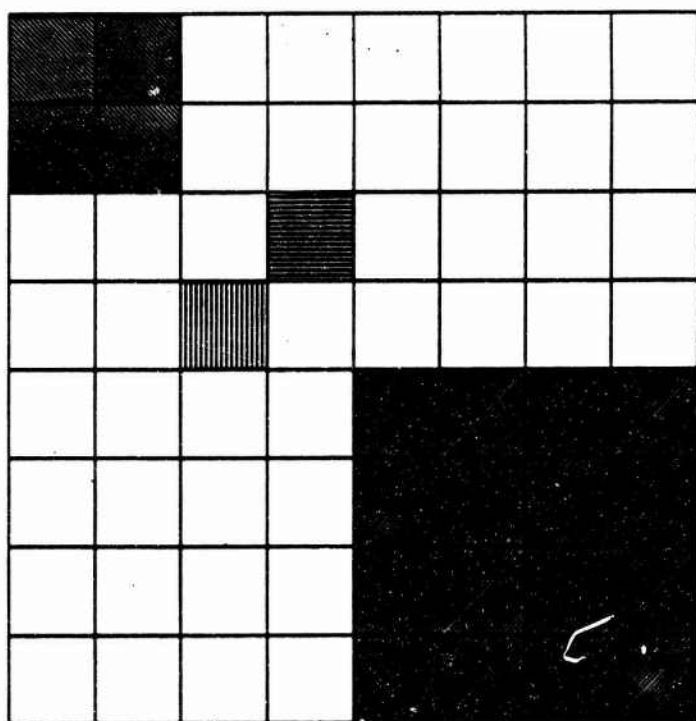Order
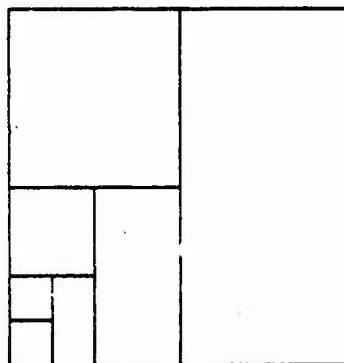
Morton
Order

Figure 2.6
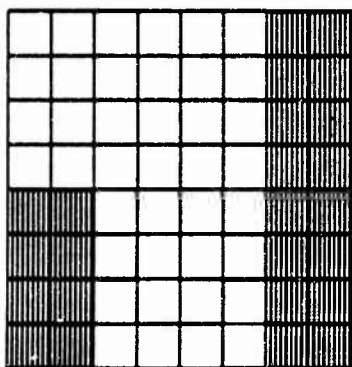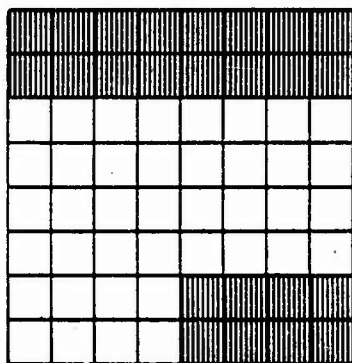
Figure 2.7
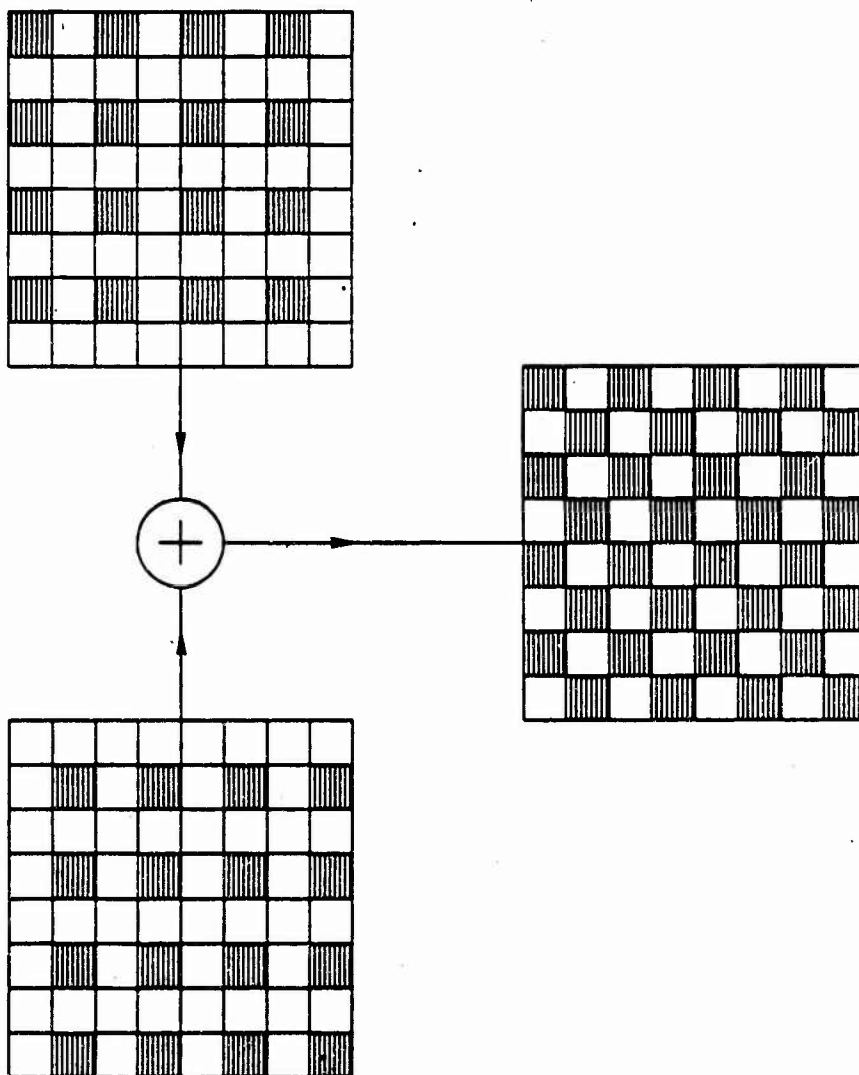
Figure 2.8
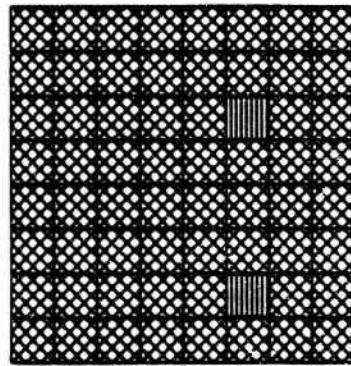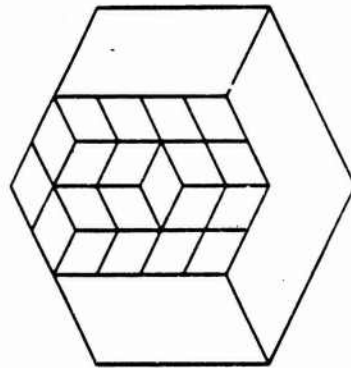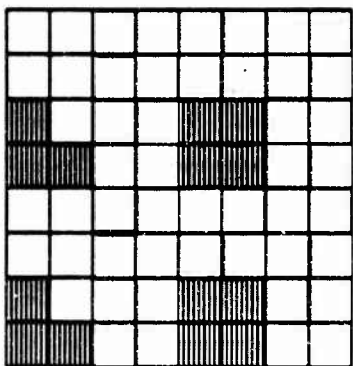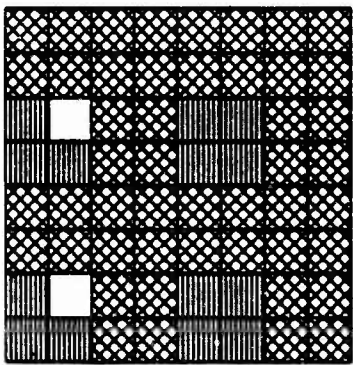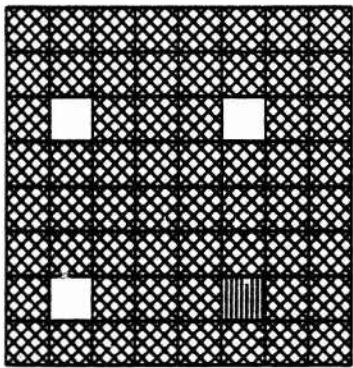
Figure 2.9
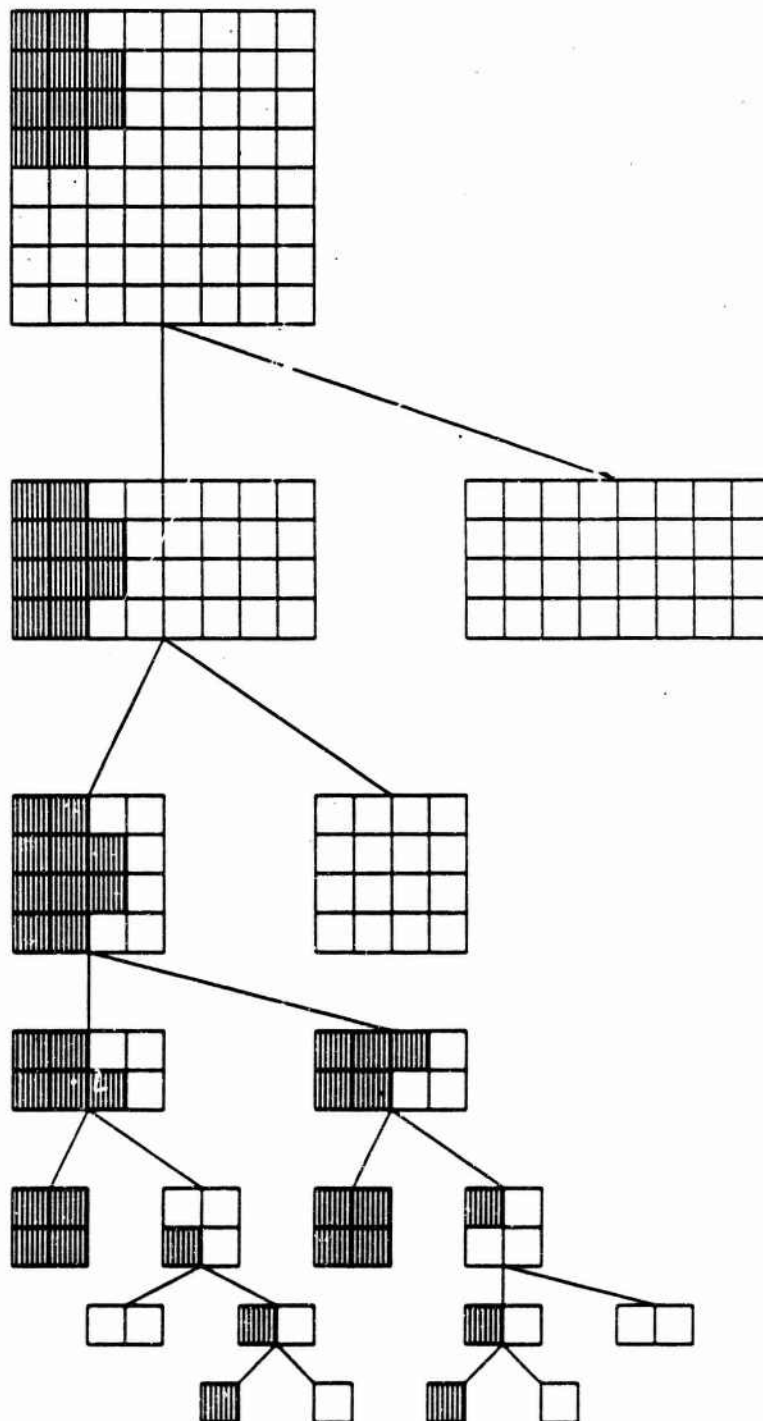
Figure 2.10
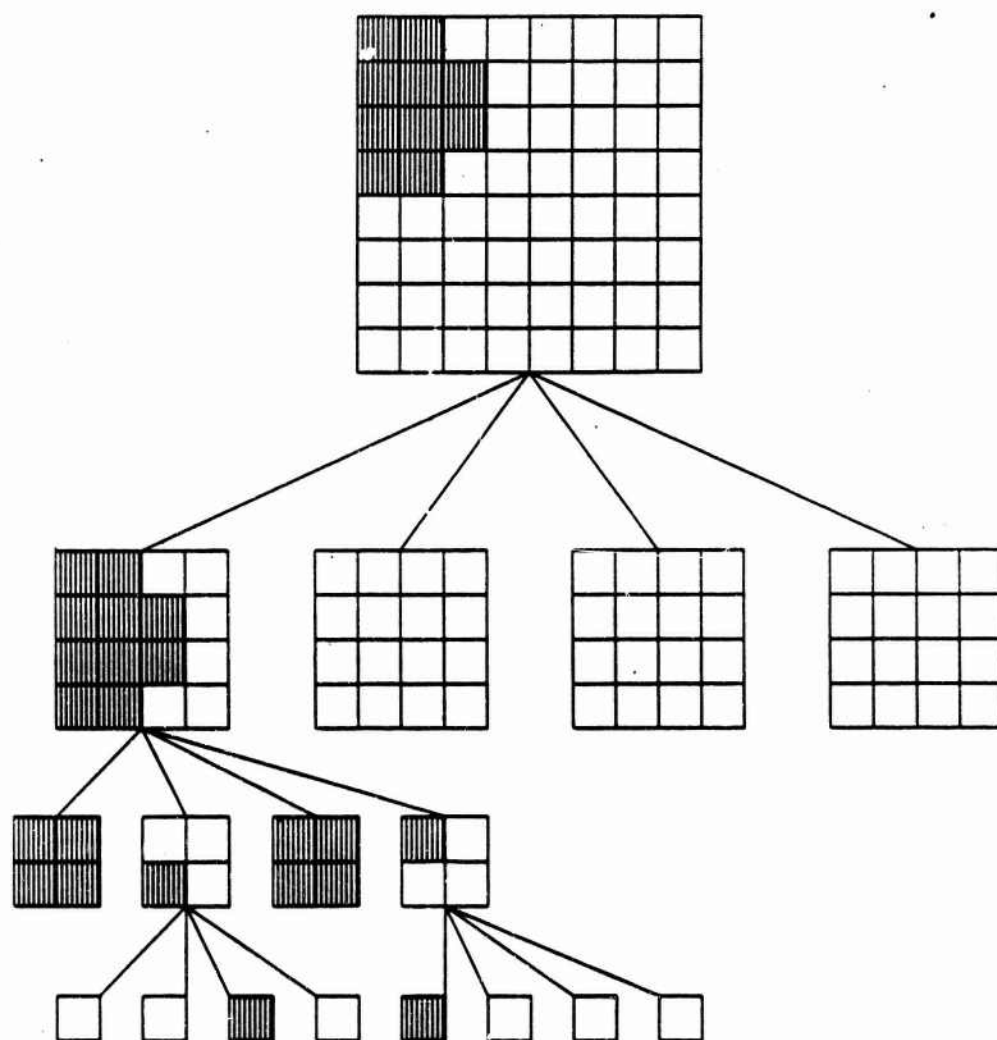
Figure 2.11
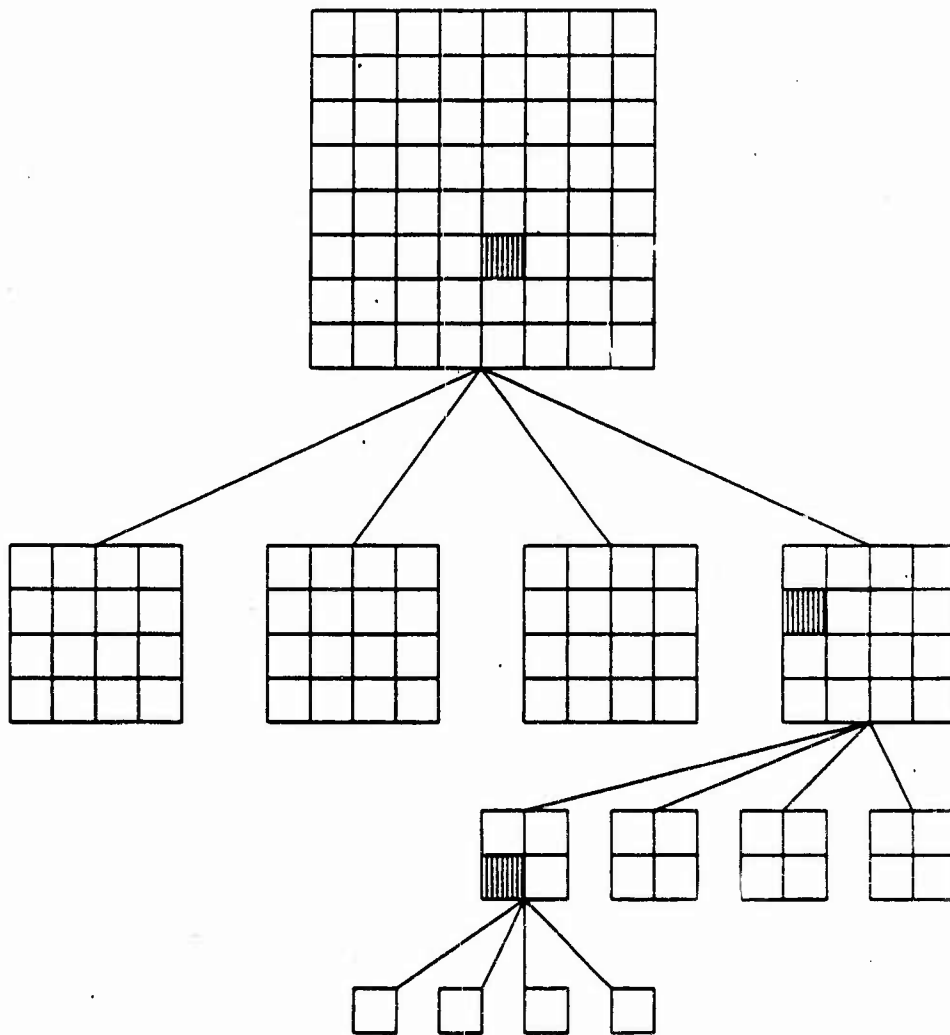
Figure 2.12

Figure 2.13

Figure 2.13

D RAS00.MAC D RAS01.MAC D RAS02.MAC D RAS03.MAC

D RAS04.MAC D RAS05.MAC D RAS06.MAC D RAS07.MAC

D RAS08.MAC D RAS09.MAC D RAS10.MAC D RAS11.MAC

D_RAS12.MAC D_RAS13.MAC D_RAS14.MAC D_RAS15.MAC P3

Figure 2.14

Address bit:0 Count: 2   Address bit:1 Count: 0   Address bit:2 Count: 0   Address bit:3 Count: 0

D PEA00.MAC   D PEA01.MAC   D PEA02.MAC   D PEA03.MAC

Address bit:4 Count: 0   Address bit:5 Count: 0   Address bit:6 Count: 0   Address bit:7 Count: 0

D PEA04.MAC   D PEA05.MAC   D PEA06.MAC   D PEA07.MAC

Address bit:8 Count: 0   Address bit:9 Count: 0   Address bit:10 Count: 0   Address bit:11 Count: 0

D PEA08.MAC   D PEA09.MAC   D PEA10.MAC   D PEA11.MAC

Address bit:12 Count: 0   Address bit:13 Count: 0   Address bit:14 Count: 0   Address bit:15 Count: 0

D_PEA12.MAC   D_PEA13.MAC   D_PEA14.MAC   D_PEA15.MAC   P2

Figure 2.15

D GP00.MAC D GP01.MAC D GP02.MAC D_GP03.MAC

D GP04.MAC D GP05.MAC D GP06.MAC D GP07.MAC

D GP08.MAC D GP09.MAC D GP10.MAC D GP11.MAC

D_GP12.MAC D_GP13.MAC D_GP14.MAC D_GP15.MAC   P1

SLM

POHM

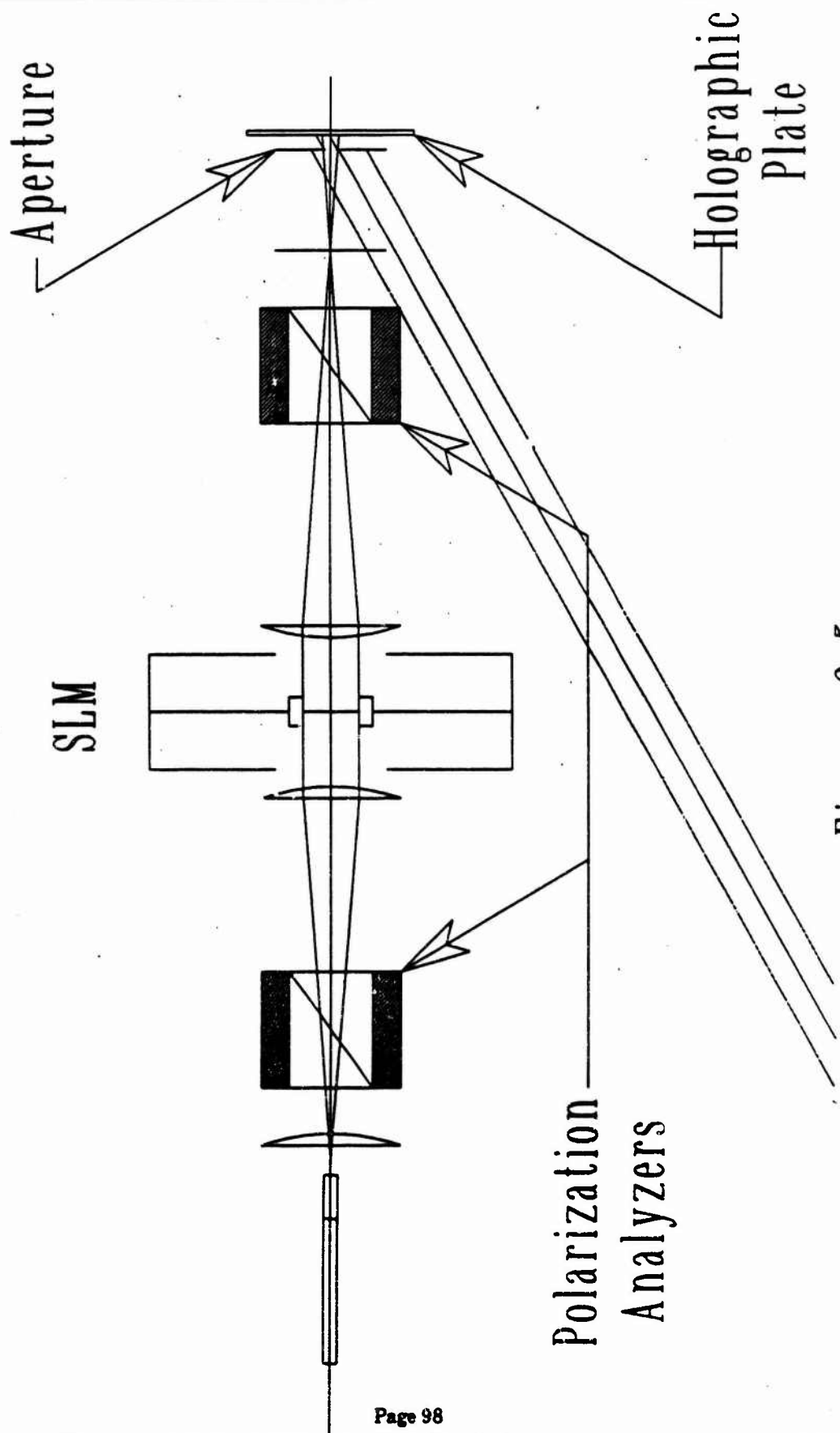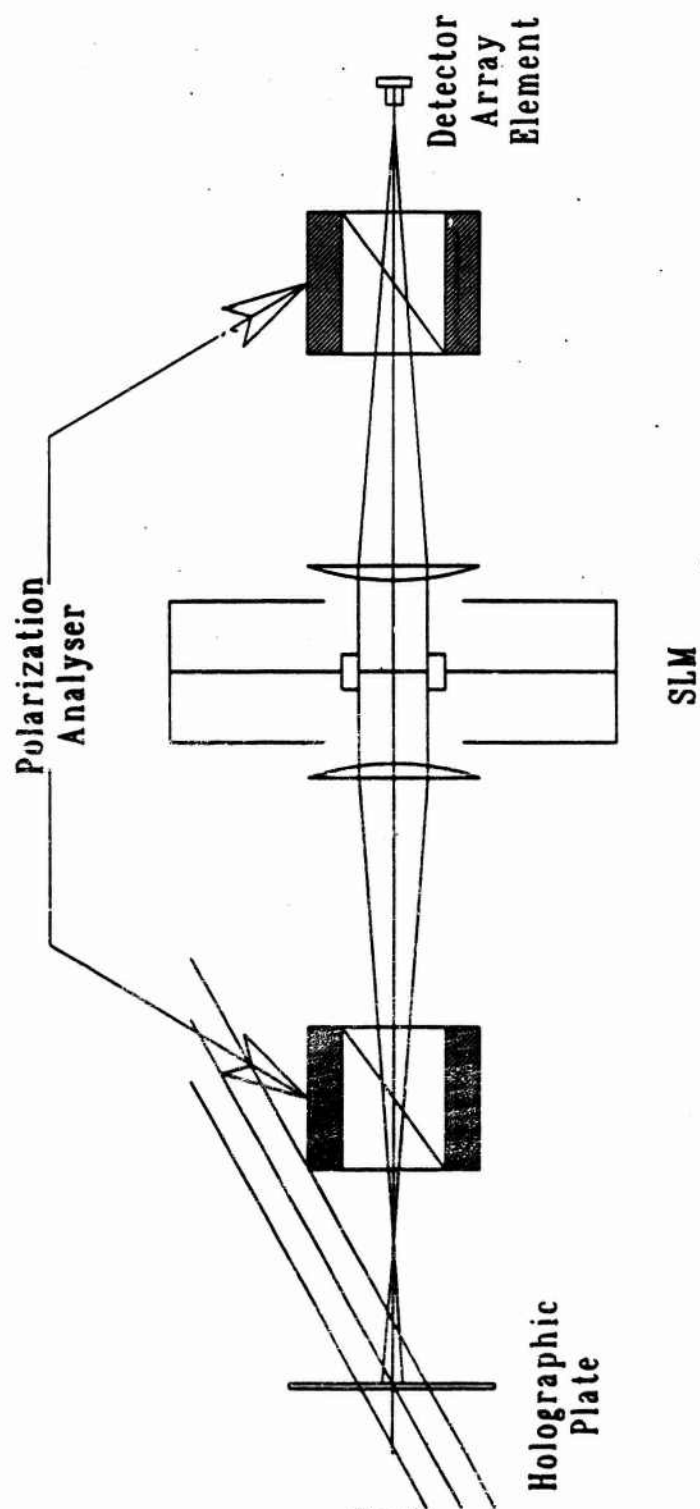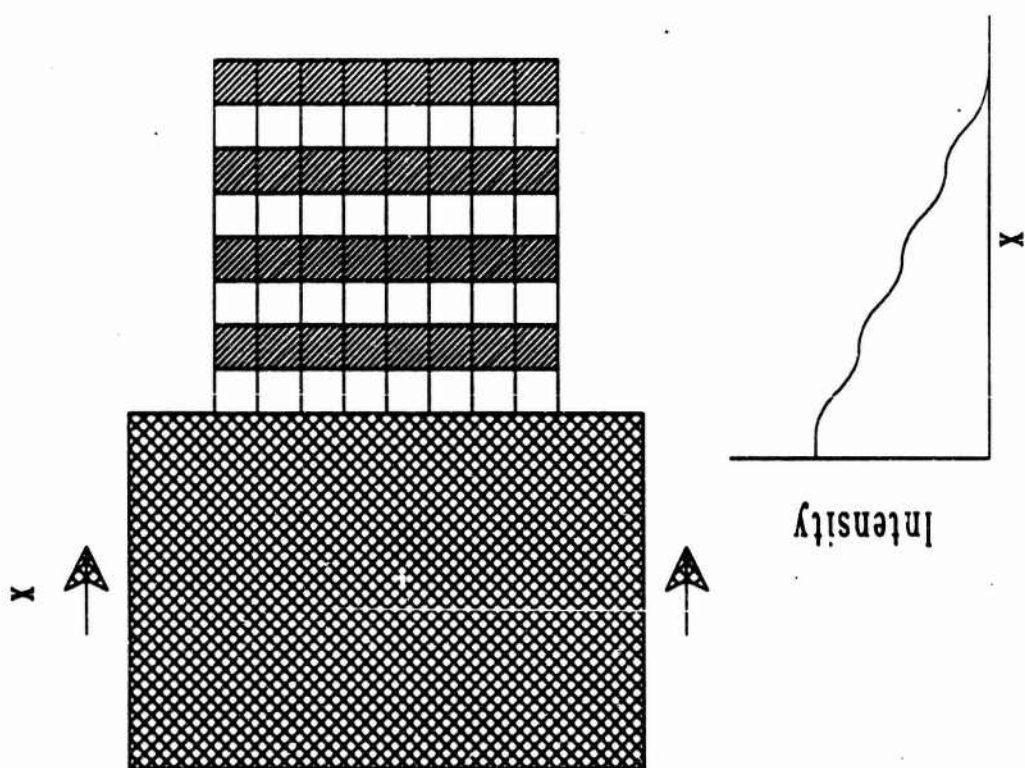Inverting
OSLM

Figure 2.16

Figure 2.17

Figure 2.18
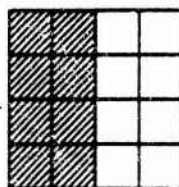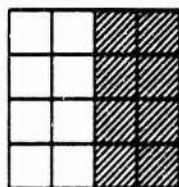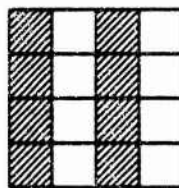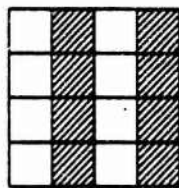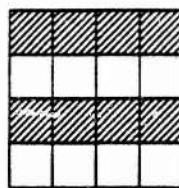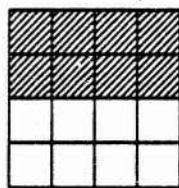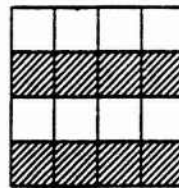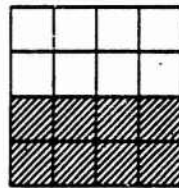
Figure 2.19
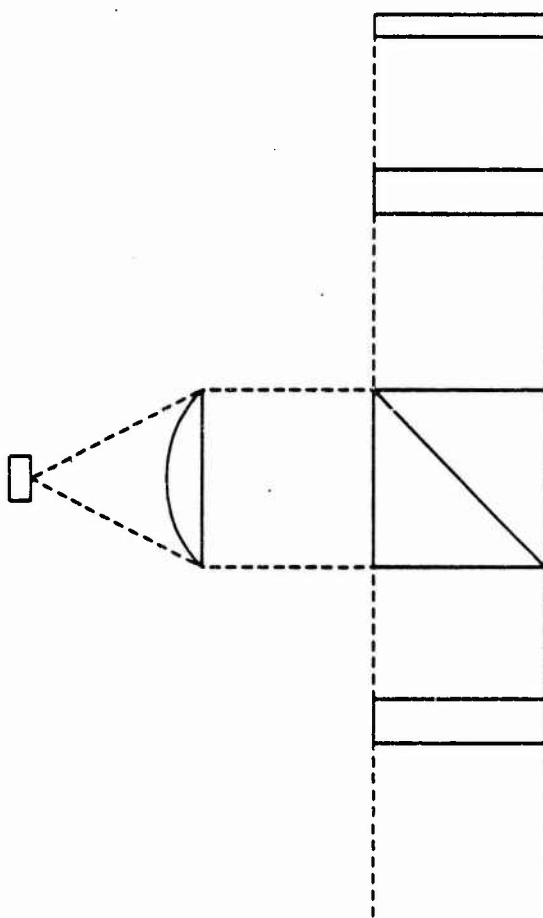
Figure 2.20

Figure 2.21
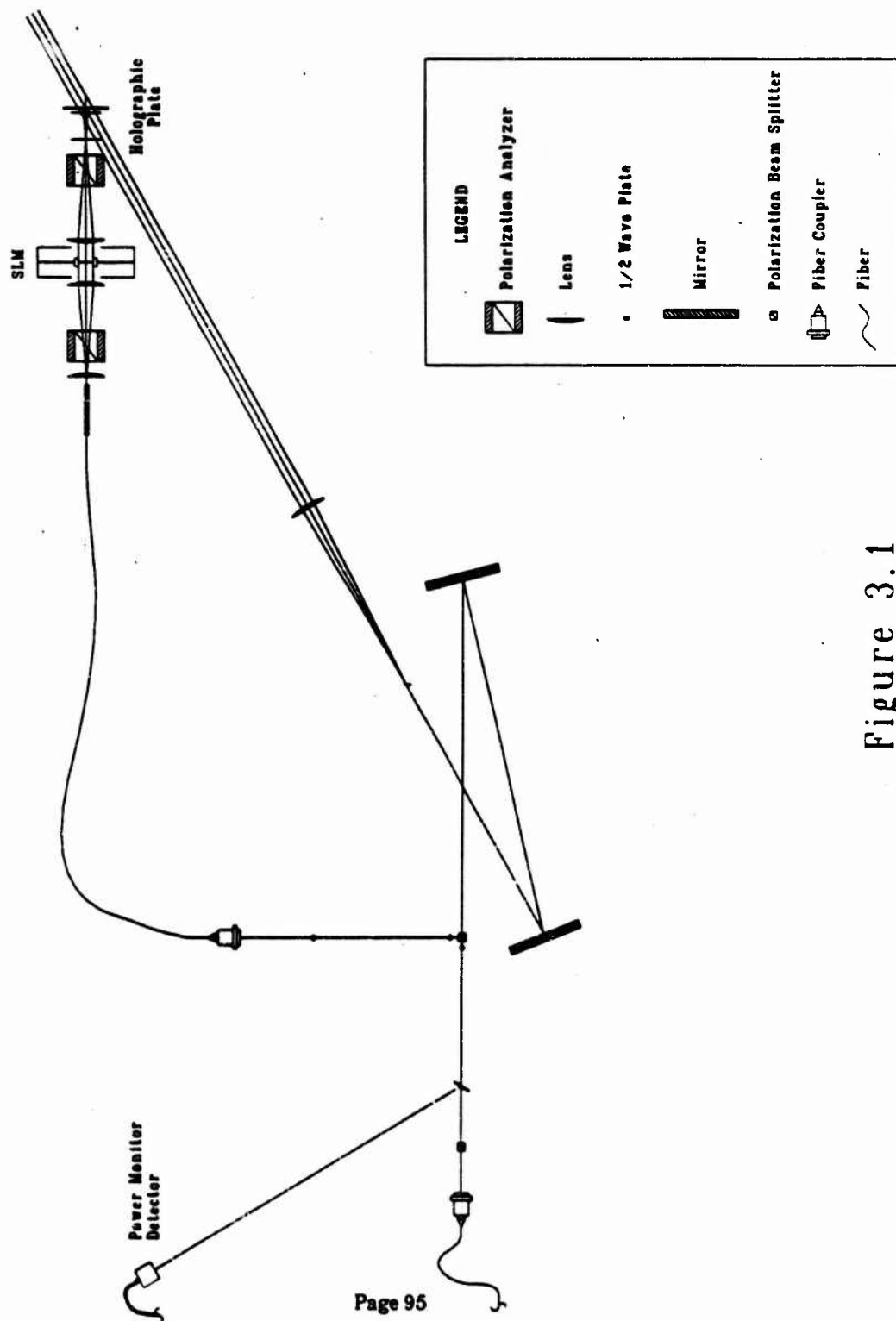
Figure 3.1
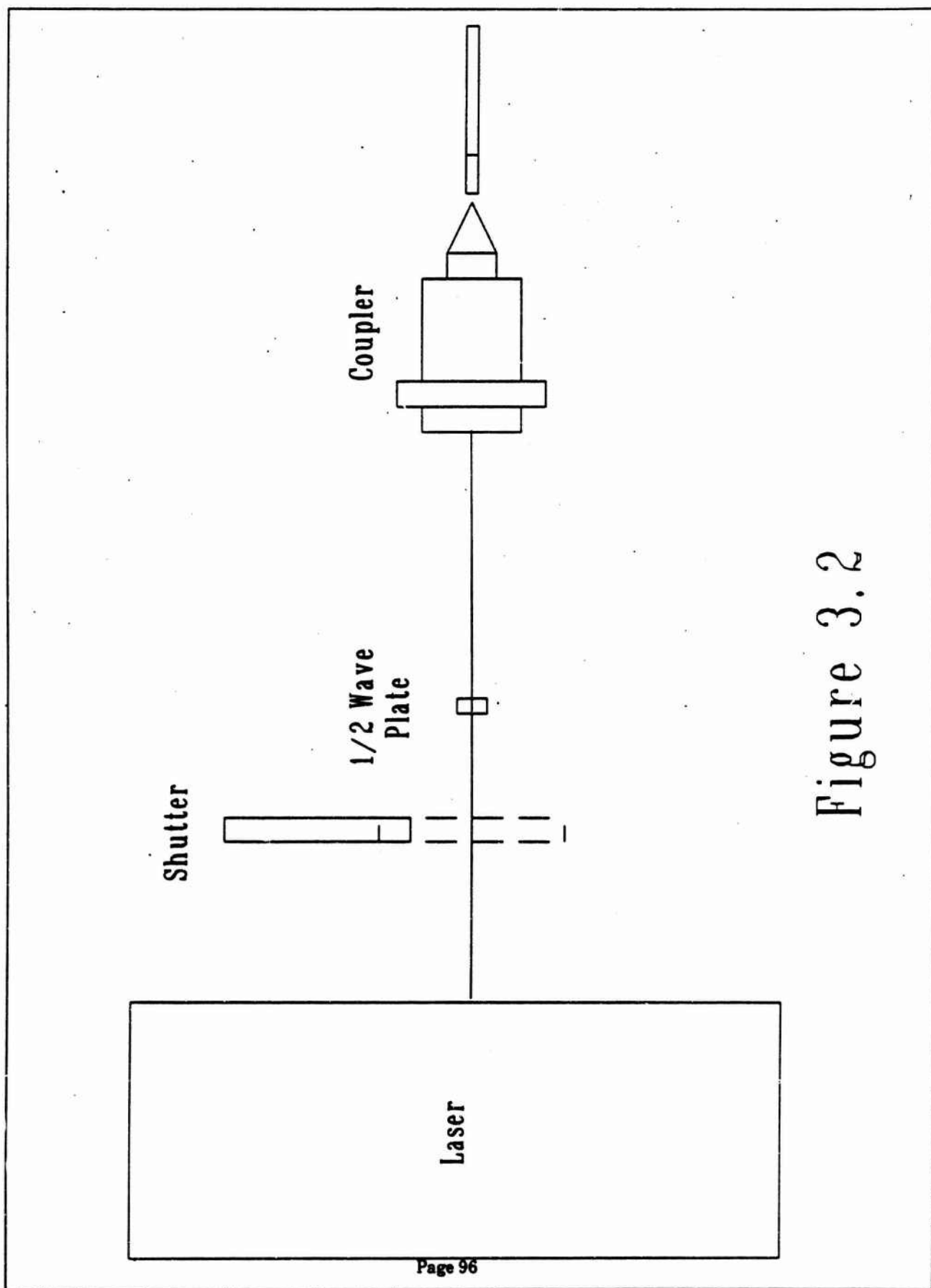
Figure 3.2

Figure 3.4

Figure 3.5

Detector
Array
Element

Polarization
Analyser

SLM

Holographic
Plate

Page 99
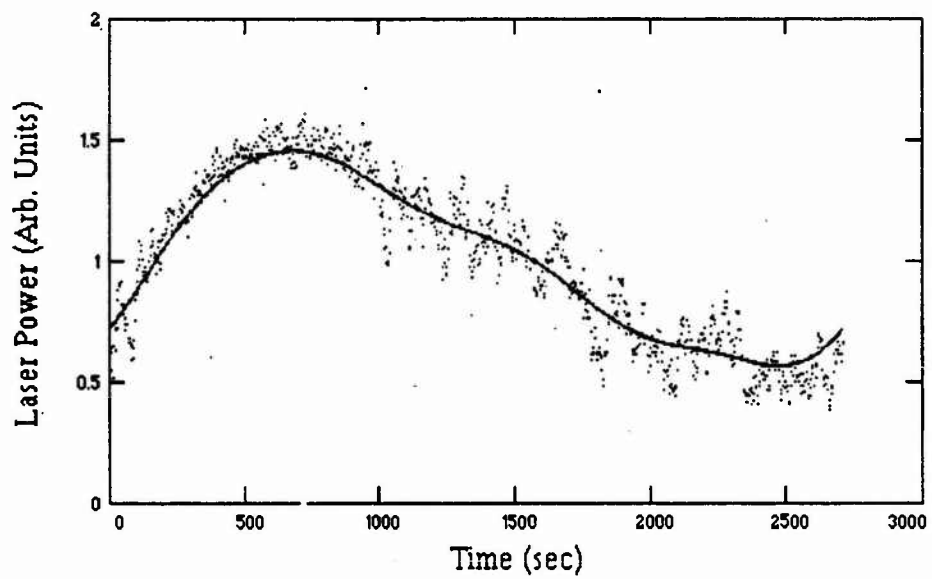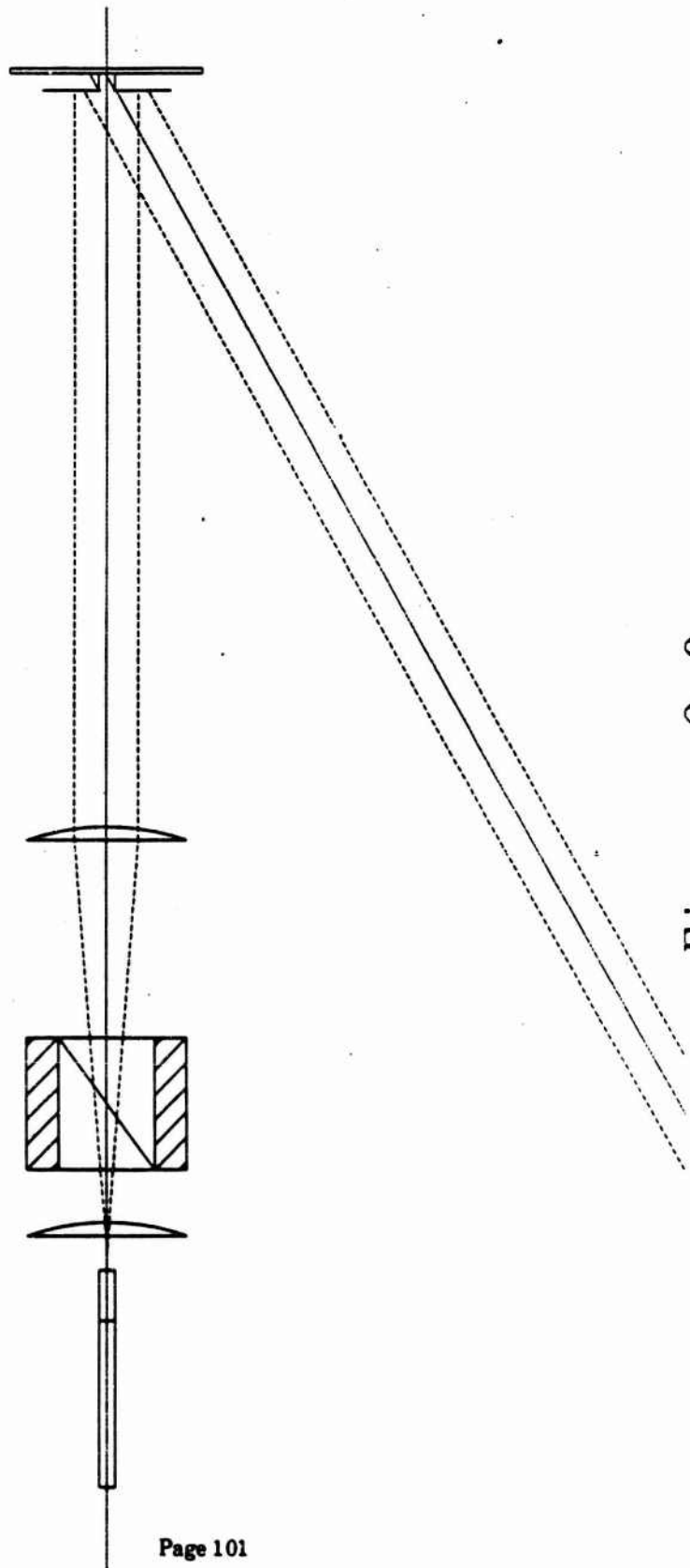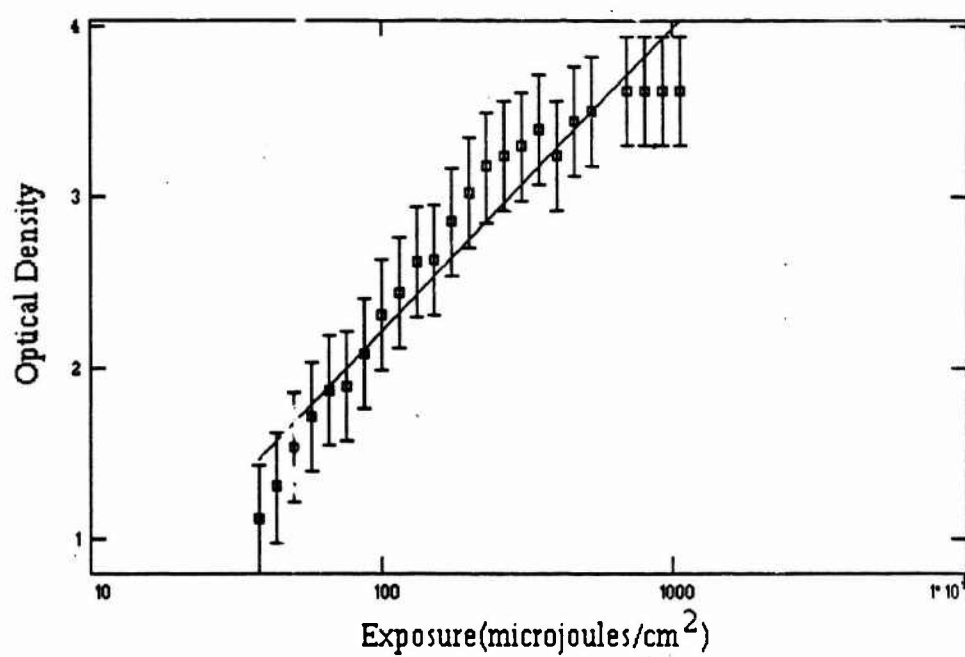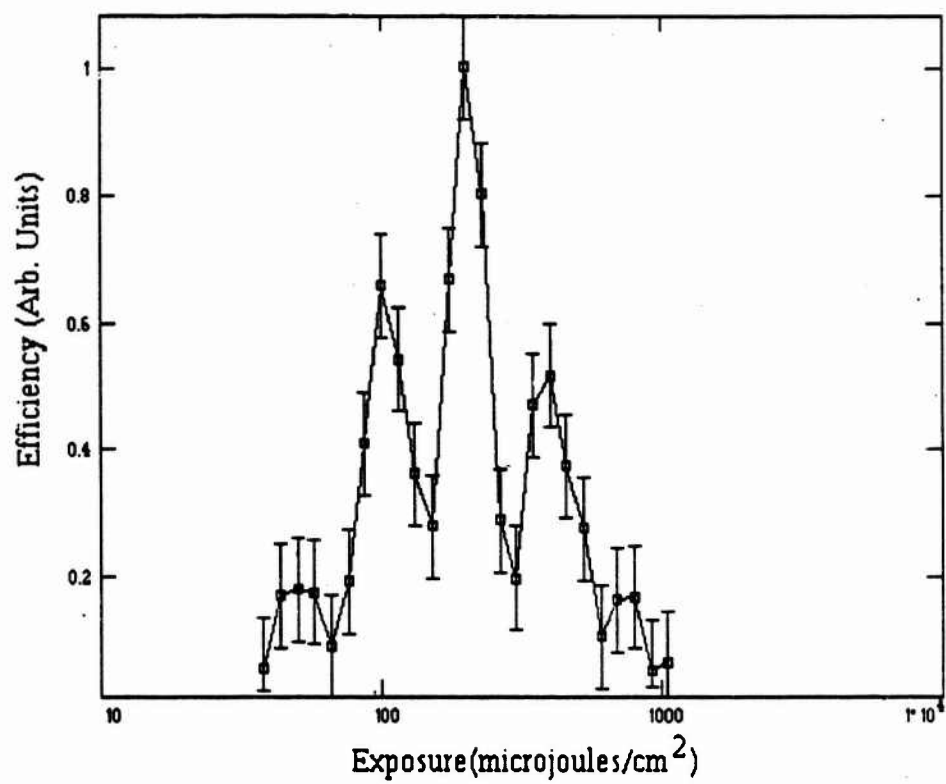
Figure 3.6

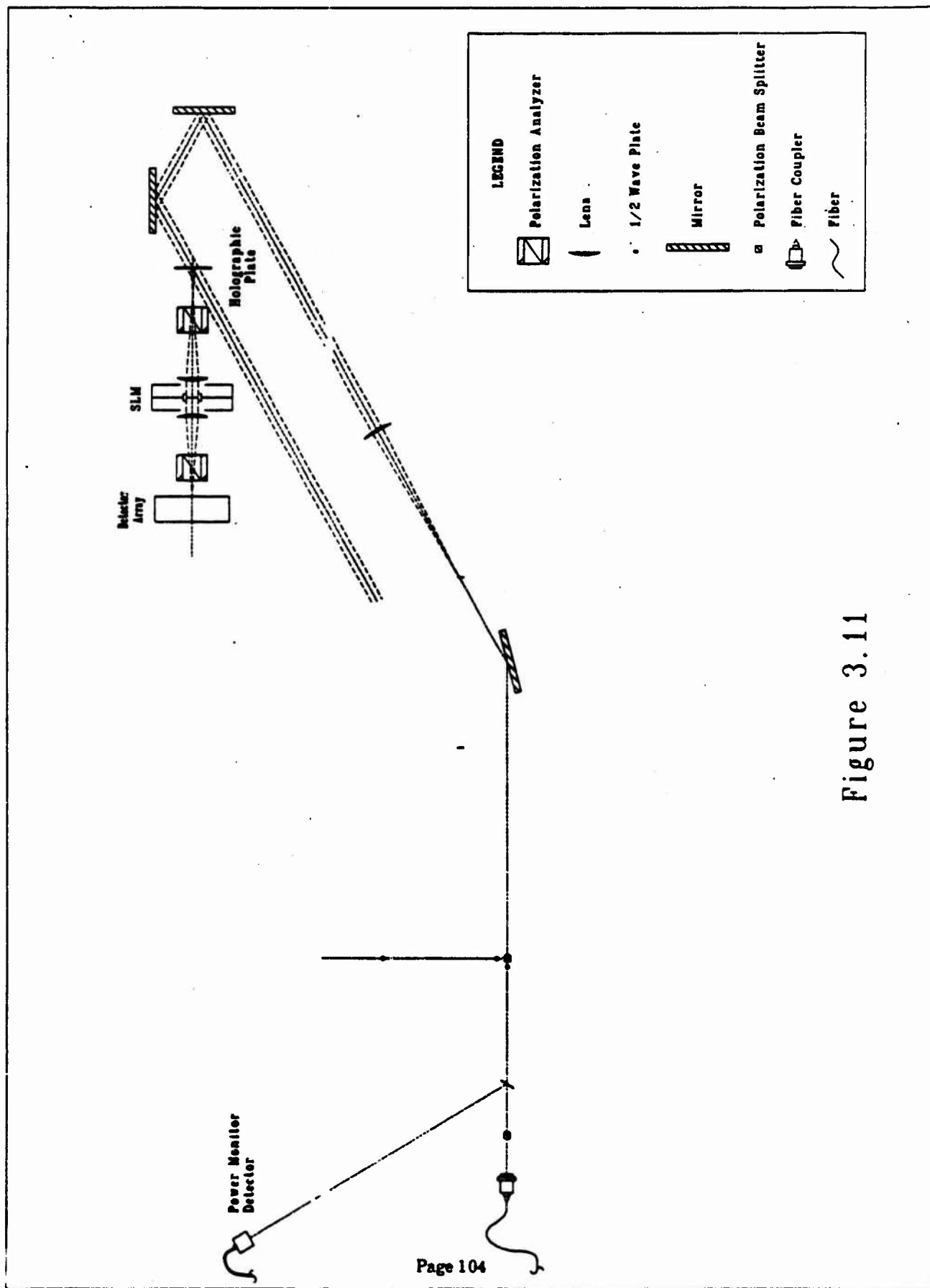Figure 3.7

Figure 3.8

Figure 3.9

Figure 3.10

Figure 3.11

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESC Program Offices (POs) and other ESC elements to perform effective acquisition of C3I systems. In addition, Rome Laboratory's technology supports other AFMC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.